Cours PHP Accéléré

Version 0.9.1

Gérard Rozsavolgyi

septembre 19, 2018

Table des matières

CHAPITRE 1

Ce cours au format epub

PHP en accéléré format epub

CHAPITRE 2

Ce cours au format pdf

PHP en accéléré en pdf

CHAPITRE 3

Tables des matières :

3.1 Caractéristiques Principales

3.1.1 Langage interprété

— Pas de compilation
— Exécuté instruction par instruction
— Multi-plateformes
— Compilateur AOT/ByteCode en PHP7 Zend
— Compilateur JIT pour HHVM Facebook
— Langage Hack proposé par Facebook

3.1.2 Spécialisé dans la génération de texte ou de documents

— HTML
— PDF
— Images

3.1.3 Fichiers d'extension .php

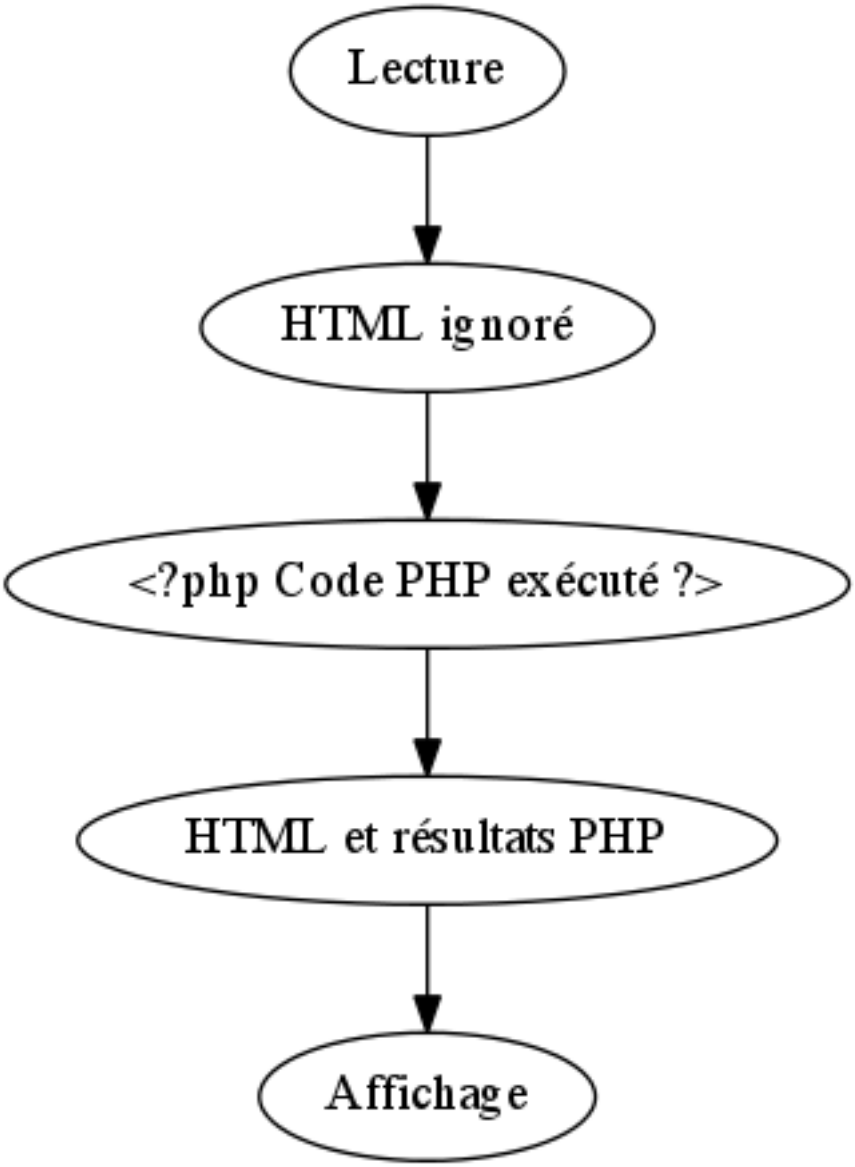
— Code inséré dans une page HTML
— Entre balises <?php et ?>

3.2 Fonctionnement

3.2.1 L'interpréteur

lit un fichier source .php puis génère un flux de sortie avec les règles suivantes :

- toute ligne située à l'extérieur d'un bloc PHP (entre <?php et ?>) est recopiée inchangée dans le flux de sortie
- le code PHP est interprété et génère éventuellement des résultats intégrés eux aussi au flux de sortie
- les erreurs éventuelles donnent lieu à des messages d'erreurs qu'on retrouve également dans le flux de sortie (selon la configuration du serveur)
- une page html pure sauvegardée avec l'extension .php sera donc non modifiée et renvoyée telle quelle ...



3.3 Historique

3.3.1 Créateur

Rasmus Lerdorf, un programmeur Groenlandais avec la nationalité canadienne, crée PHP en 1994 pour analyser les connexions sur son site Web. Il réalise les 2 premières moutures du langage (v1 et v2). En 1997, deux étudiants, Andi Gutmans et Zeev Suraski, reprennent le moteur, il en sortira PHP 3.0 puis les outils Zend.

Note : Le langage PHP a subi de profonds remaniements et a bénéficié de beaucoup d'enrichissements depuis ces premières versions. La première version objet de PHP (la version 4) a été profondément remaniée lors du passage de PHP4.0 à PHP5.0 et s'inspire désormais largement du modèle de Java.

La version actuelle de PHP est la 7.2, sortie en 2018. La version 7 est sortie en Décembre 2015 et il n'y a jamais eu de version 6 ! PHP 7 (https ://wiki.php.net/rfc/php7timeline)

Avertissement : Le début du développement de la version 6 de PHP date de 2005 et a été abandonnée en raison de difficultés d'intégration du support Unicode. Les autres fonctionnalités envisagées ont depuis été intégrées à PHP 5.3 ou 5.4. Ca n'a pas empêché un certain nombre de livres intitulés PHP 6 de sortir bien que cette version n'existera jamais ...

3.3.2 La saga PHP 7 :

- Facebook a publié en 2011 HipHop Virtual Machine dit *HHVM*, une machine virtuelle permettant de pré-compiler le code PHP en bytecode à la manière de Java (JIT Compiler).
- En 2014, Facebook sort le langage Hack, s'exécutant grace à HHVM et apportant des fonctionnalités comme le typage fort, des classes paramétrables, une API de collections plus riche et cohérente, des traitements asynchrones et davantage de sécurité avec l'extension XHP.
- Le framework PHP Phalcon (extension PECL de PHP), propose Zephir (Zend Engine PHp Intermediate) (https ://github.com/phalcon/zephir) qui permet la création d'extensions rapides en PHP (codées en C) de manière simplifiée (sans écrire du C directement).
- La société Zend a réagi face à HHVM en accélérant le développement de la nouvelle mouture de PHP 7 et en améliorant ses performances avec un mécanisme de compilation AOT (Ahead of Time)

3.3.3 PHP

Signifie d'abord *Personal Home Pages* puis *HypertextPreProcessor*

3.3.4 Syntaxe et structure

— Proche du C ou du Perl
— Peu structuré au début
— Possibilité de définir des fonctions et des classes
— Très bonnes performances pour PHP 7 (améliorations de l'ordre de 50% de la vitesse d'exécution)

3.3.5 Utilisation

— Grand succès
— Utilisation par de très grands sites
— beaucoup de code libre disponible.
— des dizaines de millions de sites Web l'utilisent à travers le monde...
— Comme Le Monde (http ://lemonde.fr/) , Facebook (http ://facebook.fr/) ou Yahoo (http ://yahoo.fr/)

3.3.6 CMS

Les grands CMS Content Management Systems ou Systèmes de Gestion de Contenus utilisent PHP, comme :
— Wordpress
— Joomla
— Drupal

Les CMS de ecommerce aussi :

— Prestashop
— Magento

3.3.7 Frameworks

De grands Frameworks de développement Web aussi sont disponibles en PHP :
— Symfony
— Zend
— Laravel
— Phalcon
— CakePHP
— Yii
— Slim

Note : Plus de 300 millions de sites sont réalisés en PHP à travers le monde !

3.4 Installation PHP

Indication : Tout informaticien doit savoir ce que LAMP veut dire . . .

3.4.1 LAMP :

— Linux
— Apache
— MySQL
— PHP

Par exension, le logiciel équivalent pour Windows, s'est retrouvé nommé :

3.4.2 WAMP :

Pour Windows

Et MAMP pour les Macs. . .

3.4.3 MAMP :

Pour Mac. Pas indispensable car Apache et PHP sont installés sur Mac OS 10.xx mais un peu plus simple à configurer. Il vous faudra alors installer MySQL soit dans un paquet soit en utilisant *homebrew* ou *macports*

Pour toutes les plate-formes, on peut aussi installer XAMPP (https ://www.apachefriends.org/fr/index.html)

Revenons à l'OS de référence à présent.

3.4.4 Sous Linux :

— Installer Apache2
— Installer PHP7
— Télécharger éventuellement la documentation (paquet php-doc)
— Vérifier le fichier php.ini
— Installer MySQL (client et serveur)
— Installer PHPMyAdmin
— Installer des paquets complémentaires (dont les noms commencent par php5-)

3.5 Configuration PHP

3.5.1 Le fichier PHP.ini

Le fichier PHP.ini contient toutes les directives essentielles de réglage.

— Taille des fichiers téléchargeables
— Safe-Mode
— Affichage et traitement des erreurs
— Communication avec MySQL

Danger : Attention, les directives de ce fichier sont très importantes pour la sécurité d'un serveur en production. Il faudra y veiller et les vérifier minutieusement dans ce cas. Sous certaines distributions de Linux, il existe 2 versions de ce fichier une de developpement et une autre pour un serveur de production. N'oubliez pas d'activer la bonne version selon le contexte et de la vérifier en tous les cas.

3.5.2 Les directives principales PHP.ini :

Ces directives sont très nombreuses. J'ai retenu les plus importantes dans le fichier suivant en commentant leur rôle.

```
[PHP]

;;;;;;;;;;;;;;;;;;;
; About php.ini   ;
;;;;;;;;;;;;;;;;;;;
; Fichier de configuration principal de PHP
; qui permet de préciser les principales options
; Sous certaines distributions Linux, il en existe 2 versions:
; une de developpement et une autre pour un serveur de production

;;;;;;;;;;;;;;;;;;;;
; Language Options ;
;;;;;;;;;;;;;;;;;;;;

; Pour activer PHP
engine = On

; On peut mettre à faux : les tags <? .... ?> ne sont pas reconnus.
short_open_tag = Off

; Allow ASP-style <% %> tags.
; http://php.net/asp-tags
asp_tags = Off

; The number of significant digits displayed in floating point␣
↪numbers.
```

(suite sur la page suivante)

(suite de la page précédente)

```
; http://php.net/precision
precision = 14

; Compatibilité an 2000
y2k_compliance = On

; Taille des buffers
output_buffering = 4096

; Pour servir ou pas des pages compressées
zlib.output_compression = Off

; Mettre à On pour forcer les flush en phase de debuggage
implicit_flush = Off


; Safe Mode
; http://php.net/safe-mode
; On peut le laisser désactivé car
; a été déclaré OBSOLETE depuis PHP 5.3
Safe_mode = Off


;Pour désactiver certaines fonctions PHP
; indépendant du safe_mode
; http://php.net/disable-functions
disable_functions =

; meme chose avec des classes
disable_classes =

; Colors for Syntax Highlighting mode.
; A utiliser avec la fonction highlight_file() = show_source()
Highlight.string  = #DD0000
highlight.comment = #FF9900
highlight.keyword = #007700
highlight.bg      = #FFFFFF
highlight.default = #0000BB
highlight.html    = #000000

;;;;;;;;;;;;;;;;;;
; Miscellaneous ;
;;;;;;;;;;;;;;;;;;

; On peut l'enlever sur un serveur de production
; mais n'est pas une menace de sécurité
expose_php = On

;;;;;;;;;;;;;;;;;;;
```

(suite sur la page suivante)

(suite de la page précédente)

```
; Resource Limits ;
;;;;;;;;;;;;;;;;;;;

;Temps d'éxécution max d'un script
;Attention si vous avez du code un peu long à s'éxécuter !
max_execution_time = 30

; Traitement des données reçues
; laisser la valeur par défaut
max_input_time = 60

; Taille mémoire maxi donnée à un script PHP
memory_limit = 128M

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Error handling and logging ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Pour un serveur de Production: E_ALL & ~E_DEPRECATED
; Pour un serveur de développement
error_reporting = E_ALL | E_STRICT

; Affichage des erreurs sur la sortie standard
; cad sur le navigateur
; A désactiver sur un serveur de production
; Utile pour un développeur
display_errors = On

; Affichage des erreurs au démarrage de PHP
; Pour débugger des erreurs sur des plugins
; ou des modules complémentaires de PHP
display_startup_errors = Off

; logger les erreurs
; A activer
log_errors = On

; Set maximum length of log_errors. In error_log information about␣
 ↪the source is
; added. The default is 1024 and 0 allows to not apply any maximum␣
 ↪length at all.
; http://php.net/log-errors-max-len
log_errors_max_len = 1024

;Ne pas répéter les erreurs identiques
ignore_repeated_errors = On

; ... sauf si elles proviennent de fichiers différents
ignore_repeated_source = Off
```

(suite sur la page suivante)

(suite de la page précédente)

```
; Rapporter les fuites de mémoire
; A activer en phase de dévelopement
report_memleaks = On

; La variable $php_errormsg
; contiendra le texte du dernier message
; d'erreur
; A désactiver sur un serveur de production
track_errors = On

; http://php.net/html-errors
html_errors = On

; A faire pointer sur une copie locale de la documentation
; de PHP
; A désactiver sur un serveur de production
docref_root = "/docs/php/"

; Extension des fichiers de documentation
docref_ext = .html

; Chaine à afficher avant un message d'erreur
; Ici pour qu'il s'affiche en rouge
; Réservé aux serveurs de dévelopement
error_prepend_string = "<font color=#ff0000>"

; Fermeture du tag précédent
error_append_string = "</font>"

; Pour changer le fichier où sont loggées
; les erreurs. Laisser inchangé sauf
; cas particulier
;error_log = syslog

;;;;;;;;;;;;;;;;;
; Data Handling ;
;;;;;;;;;;;;;;;;;

; The separator used in PHP generated URLs to separate arguments.
; PHP's default setting is "&".
; http://php.net/arg-separator.output
; Example:
;arg_separator.output = "&amp;"

; List of separator(s) used by PHP to parse input URLs into␣
↪variables.
; PHP's default setting is "&".
; NOTE: Every character in this directive is considered as␣
↪separator!
```

(suite sur la page suivante)

(suite de la page précédente)

```
; http://php.net/arg-separator.input
; Example:
;arg_separator.input = ";&"

; This directive determines which super global arrays are␣
↪registered when PHP
; starts up. If the register_globals directive is enabled, it also␣
↪determines
; what order variables are populated into the global space. G,P,C,E␣
↪& S are
; abbreviations for the following respective super globals: GET,␣
↪POST, COOKIE,
; ENV and SERVER. There is a performance penalty paid for the␣
↪registration of
; these arrays and because ENV is not as commonly used as the␣
↪others, ENV is
; is not recommended on productions servers. You can still get␣
↪access to
; the environment variables through getenv() should you need to.
; Default Value: "EGPCS"
; Development Value: "GPCS"
; Production Value: "GPCS";
; http://php.net/variables-order
variables_order = "GPCS"

; laisser la valeur par défaut
request_order = "GP"

; Ca fait longtemps qu'il faut garder cette directive à Off
register_globals = Off

; Determines whether the deprecated long $HTTP_*_VARS type␣
↪predefined variables
; are registered by PHP or not. As they are deprecated, we␣
↪obviously don't
; recommend you use them. They are on by default for compatibility␣
↪reasons but
; they are not recommended on production servers.
; Default Value: On
; Development Value: Off
; Production Value: Off
; http://php.net/register-long-arrays
register_long_arrays = Off

; A activer seulement si vous voulez utiliser PHP
; en ligne de commande et lui passer des arguments
register_argc_argv = Off

; Meilleure performance avec :
```

(suite sur la page suivante)

(suite de la page précédente)

```
auto_globals_jit = On

; Taille maximale des données acceptées en POST
; http://php.net/post-max-size
post_max_size = 8M

; A éviter désormais
magic_quotes_gpc = Off

; idem
magic_quotes_runtime = Off

; mimetype par défaut : text/html
default_mimetype = "text/html"

; Jeu de caractères par  défaut
; laisser à vide ou choisir un jeu de caractères
;default_charset = "iso-8859-1"
default_charset = "utf-8"

;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;

;
; Autoriser les "uploads" de fichiers
file_uploads = On

; Spécifier le répertoire temporaire por les fichiers
; uploadés :
;upload_tmp_dir = /tmp/upload-dir

; Taille maxi pour les fichies uploadés
upload_max_filesize = 4M

; Nbre de fichiers maxi pouvant être uploadés en une seule requête
max_file_uploads = 20

;;;;;;;;;;;;;;;;;;
; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;

; Whether to allow the treatment of URLs (like http:// or ftp://)␣
 ↪as files.
; http://php.net/allow-url-fopen
allow_url_fopen = On

; Whether to allow include/require to open URLs (like http:// or␣
 ↪ftp://) as files.
```

(suite sur la page suivante)

(suite de la page précédente)

```
; http://php.net/allow-url-include
allow_url_include = Off

; Define the anonymous ftp password (your email address). PHP's␣
↪default setting
; for this is empty.
; http://php.net/from
;from="john@doe.com"

; Define the User-Agent string. PHP's default setting for this is␣
↪empty.
; http://php.net/user-agent
;user_agent="PHP"

; Timeout pour les flux basés sur des sockets
default_socket_timeout = 60

;;;;;;;;;;;;;;;;;;;;;
; Dynamic Extensions ;
;;;;;;;;;;;;;;;;;;;;;

; ; Sous Windows:
;   extension=msql.dll
; ... et sous UNIX:
;
;   extension=msql.so
;
; ... ou avec un chemin:
;
;   extension=/path/to/extension/msql.so
;

;;;;;;;;;;;;;;;;;;;
; Module Settings ;
;;;;;;;;;;;;;;;;;;;

[Date]
; Fuseau horaire utilisé
date.timezone ="Europe/Paris"

[iconv]
; conversion d'un système d'encodage à un autre
;iconv.input_encoding = ISO-8859-1
;iconv.internal_encoding = ISO-8859-1
;iconv.output_encoding = ISO-8859-1

[Pdo_mysql]
; En cas d'utilisation du nouveau moteur  mysqlnd
pdo_mysql.cache_size = 2000
```

(suite sur la page suivante)

(suite de la page précédente)

```
; Socket par défaut pour la connexion à MySQL
; La valeur par défaut fonctionne le plus souvent
pdo_mysql.default_socket=/var/mysql/mysql.sock

[mail function]
; For Win32 only.
; http://php.net/smtp
SMTP = localhost
; http://php.net/smtp-port
smtp_port = 25

; Emplacement pour logger les appels à la fonction mail()
;mail.log =

[MySQL]
; Autorise les connexions persistantes
; N'apporte AUCUNE fonctionnalité supplémentaire
; Mais peut améliorer les performances
mysql.allow_persistent = On

; If mysqlnd is used: Number of cache slots for the internal result␣
↪set cache
; http://php.net/mysql.cache_size
mysql.cache_size = 2000

; Nbre maxi de liens persistants
mysql.max_persistent = -1

; Nombre maxi de liens permanents :
; -1 veut dire sans limitation
Mysql.max_links = -1

; Port par défaut de MySQL
mysql.default_port = 3306

; Laisser généralement la valeur par défaut
mysql.default_socket =

; Hôte par défaut pour mysql_connect()
mysql.default_host =

; Utilisateur  par défaut pour mysql_connect()
mysql.default_user =

; Passwd  par défaut pour mysql_connect()
; Ce n'est pas une bonne chose de garder
; le passwd ici !! obsolete
mysql.default_password =
```

(suite sur la page suivante)

(suite de la page précédente)

```
; Timeout de connexion à MySQL
mysql.connect_timeout = 60

; Mode de débuggage MySQL
mysql.trace_mode = Off

[MySQLi]
; Nbre maxi de liens persistants
mysqli.max_persistent = -1

; Autorise les connexions persistantes
; N'apporte AUCUNE fonctionnalité supplémentaire
; Mais peut améliorer les performances
mysqli.allow_persistent = On

; Maximum number of links.  -1 means no limit.
; http://php.net/mysqli.max-links
mysqli.max_links = -1

; If mysqlnd is used: Number of cache slots for the internal result␣
↪set cache
; http://php.net/mysqli.cache_size
mysqli.cache_size = 2000

; Por pour mysqli
mysqli.default_port = 3306

; Socket par défaut pour MySQLi
mysqli.default_socket = /tmp/mysql.sock

; Autorise ou interdit la reconnexion
mysqli.reconnect = Off

[mysqlnd]
; activation des statistiques de mysqlnd
; a des fins de réglages du serveur de BD
mysqlnd.collect_statistics = On

; Même chose avec les opérations sur la mémoire
mysqlnd.collect_memory_statistics = Off

; Size of a pre-allocated buffer used when sending commands to␣
↪MySQL in bytes.
; http://php.net/mysqlnd.net_cmd_buffer_size
;mysqlnd.net_cmd_buffer_size = 2048

; Size of a pre-allocated buffer used for reading data sent by the␣
↪server in
```

(suite sur la page suivante)

(suite de la page précédente)

```
; bytes.
; http://php.net/mysqlnd.net_read_buffer_size
;mysqlnd.net_read_buffer_size = 32768

[bcmath]
; Number of decimal digits for all bcmath functions.
; http://php.net/bcmath.scale
bcmath.scale = 0

[Session]
; .../...

; Les sessions doivent-elles utiliser les cookies ?
session.use_cookies = 1

; Envoyer les cookies à travers
; des connexions sécurisées
; le défaut est Off
;Session.cookie_secure =

; PHP maintient un cookie avec l'identifiant de session
; c'est une précaution visant à éviter
; le vol de session
; ce n'est pas une parade absolue
session.use_only_cookies = 1

; Nom de la session
session.name = PHPSESSID

; Démarrage automatique de session
; Désactivé par défaut
session.auto_start = 0

; Durée de vie du cookie
; Si placé à 0, le temps que le navigateur
; soit redémarré
session.cookie_lifetime = 0

; Domaine de validité du cookie
session.cookie_domain =

; Pour interdire à javascript d'accéder à ce cookie
session.cookie_httponly = On

;
; HTTP_REFERER doit contenir cette sous-chaine
; pour être considéré comme valide
Session.referer_check =
```

(suite sur la page suivante)

(suite de la page précédente)

```
; Durée d'expiration du document en minutes
session.cache_expire = 180

; Choix d'une fonction de hachage pour les sessions
; comme :
;   0  (MD5 128 bits)
;   1  (SHA-1 160 bits)
session.hash_function = 0

[Assertion]
; Assertions actives (défaut)
assert.active = On

; Emettre un warning en cas d'assertion non vérifiée
assert.warning = On

; S'arrêter en cas d'assertion non satisfaite
; Désactivé par défaut
;assert.bail = Off

; Fonction utilisateur à appeller en cas d'assertion non satisfaite
;assert.callback = 0
```

3.6 Exemples simples

3.6.1 bonjour

```
<!doctype html>
<html>
<head>
<meta charset="utf-8" />
<title> Bonjour depuis PHP </title>
</head>
<body>
<?php echo 'Bonjour généré dynamiquement en PHP !'; ?>
</body>
</html>
```

3.6.2 Résultat brut html

```
<!doctype html>
<html>
<head>
```

(suite sur la page suivante)

(suite de la page précédente)

```
<meta charset="utf-8" />
<title> Bonjour depuis PHP </title>
</head>
<body>
Bonjour généré dynamiquement en PHP !</body>
</html>
```

3.6.3 Exécution

bonjour

3.6.4 Infos PHP

```
<?php phpinfo(); ?>
```

3.6.5 Exécution

infos php

3.6.6 User Agent

```
<!doctype html>
<html>
<head>
<title>
Navigateur
</title>
<body>
Les informations sur le Navigateur sont :
<?php
 echo $_SERVER['HTTP_USER_AGENT'];
 ?>
</body>
</html>
```

3.6.7 Exemple de Résultat

```
Les informations sur le Navigateur sont : Mozilla/5.0
(Macintosh; U; Intel Mac OS X 10_6_4; fr-fr) AppleWebKit/
↪533.18.1
(KHTML, like Gecko) Version/5.0.2 Safari/533.18.5
```

3.6.8 Exécution

User-Agent

Vous pouvez consulter la documentation de PHP sur :

doc php (http ://php.net/manual/fr/)

3.7 Imbrication de code HTML et PHP

On peut par exemple :

— Commencer une page HTML
— puis écrire un peu de PHP
— reprendre en HTML
— etc.

3.7.1 Exemple :

```
<!doctype html>
<html>
<head>
<title>
Navigateur
</title>
<body>
Les informations sur le Navigateur sont :
<?php
 $AGENT=$_SERVER['HTTP_USER_AGENT'];
 echo $AGENT;
 echo("\n<P>");
 if (stristr($AGENT,"MSIE")) {
          ?>
          <b>Vous semblez utiliser Internet Explorer !</b>
 <?php }
     elseif (preg_match("/Firefox/i",$AGENT))
 { ?>
         <b>Vous semblez utiliser Firefox !</b>
 <?php }
                  elseif (preg_match("/chrome/i",$AGENT))
 { ?>
```

(suite sur la page suivante)

(suite de la page précédente)

```
        <b>Vous semblez utiliser Chrome !</b>
 <?php }
 elseif (preg_match("/Safari/",$AGENT))
         { ?>
        <b>Vous semblez utiliser Safari !</b>
  <?php }
  else echo "Navigateur Inconnu !";  ?>
</body>
</html>
```

ATTENTION : ça peut vite devenir **ILLISIBLE**

3.7.2 Exécution

user-agent php

3.7.3 Remèdes :

Pour ne pas écrire ce genre de code, quelques solutions courantes :

- Utiliser des fonctions PHP
- Utiliser des Classes et Objets PHP
- Séparer les modèles des Vues
- Séparer les Modèles, les Vues et les Contrôleurs (Modèle MVC)
- Utiliser des systèmes de templates comme Twig

3.8 Un formulaire HTML et sa réponse en PHP

On code ici :

- un petit formulaire HTML
- et sa réponse en PHP

3.8.1 Formulaire HTML :

```
<!doctype html>
<html>
<head>
<meta charset="utf-8" />
<title>
Formulaire html
</title>
</head>
<body>
<form action="reponse.php" method="GET">
```

(suite sur la page suivante)

(suite de la page précédente)

```
Votre nom :<input type="text" name="nom">
Votre âge :<input type="text" name="age">
<p>
<input type=submit value="Envoyer">
</form>
</body>
</html>
```

3.8.2 Sa réponse :

```
<!doctype html>
<html>
<head>
<meta charset="utf-8" />
<title>
Test Formulaire PHP
</title>
</head>
<body>
<h1>Bonjour,  <?php echo $_GET['nom'] ?></h1>
<h2>Vous semblez avoir  <?php echo $_GET['age'] ?></h2>
   <?php
    $n = $_GET['nom'];
    $a = $_GET['age'];
    ?>
Votre nom est stocké dans la variable $n
dont le type est <?php echo gettype($n) ?>

Votre âge est stocké dans la variable <b>$a</b>
<br/> dont le type est <i><?php echo gettype($a); ?></i>
<br/> On peut la transformer en <i>integer</i> en faisant :
   <?php settype($a,"integer"); ?>
<br/>
    Type de $a :<?php echo gettype($a); ?>
</body>

</html>
```

3.8.3 Exécution :

Formulaire

3.9 Les variables en PHP

3.9.1 Déclaration simple :

```
<?php
$variable = "une variable en PHP";
// Une autre variable :
$Variable = 1000;
?>
```

3.9.2 Existence de variables, la fonction isset() :

```
<?php
$a = "une variable en PHP";
if(isset($a)) echo "la variable a existe";
unset($a);
echo "la variable a a été supprimée ...";
?>
```

3.9.3 Test de variables, la fonction empty() :

```
<?php
$a = "une variable en PHP";
if (!empty($a)) echo " La variable existe et elle n\'est␣
↪pas vide !";
?>
```

Avertissement : La fonction empty() répond vrai si la variable n'existe pas et ceci sans faire aucun warning ! En outre, avant PHP 5.5, on ne peut pas l'utiliser sur autre chose que des variables (impossible d'appeller une fonction dans l'argument qu'on lui passe)

3.9.4 Test de variables en PHP 7 avec l'opérateur *coalescent* :

L'opérateur *Null coalescent* **??** permet de simplifier certains tests d'existence de variables et d'alternatives, comme par exemple :

```
<?php

// $a non initialisée
$b = 143;
```

(suite sur la page suivante)

(suite de la page précédente)

```
echo $a ?? 3; // affiche 3
echo PHP_EOL;
echo $a ?? $b ?? 7; // affiche 143
echo PHP_EOL;
```

Ce qui permet de limiter le recours à *isset* dans de nombreuses situations comme :

```
<?php
// Récupère la valeur de $_GET['email'] et  retourne 'nobody
↪'
// si elle n'existe pas.
$mail = $_GET['email'] ?? 'nobody@null';
// Equivalent à:
$mail = isset($_GET['email']) ? $_GET['email'] :
↪'nobody@null';

// Coalescing ?? peut être chainé :
// On renvoie la première valeur définie parmi
// $_GET['email'], $_POST['email'], et 'nobody@null.com'.
$mail = $_GET['email'] ?? $_POST['email'] ?? 'nobody@null';
echo "$mail\n";
```

3.9.5 Portée des variables :

— Par défaut, toutes les variables sont **locales**
— Leur portée se réduit à la fonction ou au bloc de leur déclaration
— Pour déclarer une variable globale, on peut utiliser le tableau $_GLOBALS[ ]

```
<?php $_GLOBALS['MaVar']="Bonjour"; ?>
```

3.9.6 Constantes :

```
<?php
define("USER","TOTO");
echo USER; // Notez l'absence de $ ici
?>
```

3.10 Les chaînes en PHP

3.10.1 Les bases :

Guillemets ou Cotes :

```
<?php
$var="Hello PHP";
$machaine="le contenu de \$var est $var<br>";
echo $machaine;
//ou avec des '  ':
$mystring='le contenu de $var est '.$var;
echo $mystring;
?>
```

dont le résultat sera toujours :

```
le contenu de $var est Hello PHP
```

La concaténation :

A l'aide de **.**

La longueur d'une chaine :

```
<?php int lg=strlen($chaine); ?>
```

Accéder au caractère i d'une chaine :

```
<?php echo $chaine[i]; ?>

La chaine est traitée comme un tableau indéxé par un *entier*
La plupart des tableaux de PHP sont indéxés par des chaines...
```

Mettre en majuscules/minuscules :

— avec *strtoupper()* pour obtenir des majuscules
— avec *strtolower()* pour mettre en minuscules
— avec *ucfirst()* pour mettre en majuscule la première lettre d'une chaine
— avec *ucwords()* pour mettre en majuscule la première lettre de chaque mot dans une chaine

3.10.2 Recherche de sous-chaines ou de motifs dans une chaine :

— avec *strstr()*
— avec *stristr()*
— avec *ereg()* ou *eregi()*

Par exemple :

```
<?php
$AGENT=$_SERVER['HTTP_USER_AGENT'];
echo $AGENT;
echo("\n<P>");
if (stristr($AGENT,"MSIE"))
   echo "Vous semblez utiliser Internet Explorer !</b>";
   elseif (ereg("Firefox",$AGENT))
          echo "Vous semblez utiliser Firefox !</b>";
   elseif (eregi("chrome",$AGENT))
          echo "Vous semblez utiliser Chrome !</b>";
?>
```

Indication : Les variantes de ces fonctions comportant un i indiquent une insensibilité à la casse c'est à dire que les majuscules et minuscules sont considérées comme identiques.

Exemple : Test un peu plus complet du UserAgent :

```
<?php
function getBrowser($userAgent){
  if (preg_match("/MSIE(.{5})/i",$userAgent,$num))
    return $num[0];
  elseif (preg_match("/Firefox(.*)/i",$userAgent,$num))
    return $num[0];
  elseif (preg_match("/chrome(.{4})/i",$userAgent,$num))
    return $num[0];
  elseif (preg_match("/safari/i",$userAgent,$num)){
    preg_match("/Version(.{4})/",$userAgent,$num);
    return "Safari ".$num[0];
}
  else return "Navigateur Inconnu";
}
if (!empty($_SERVER['HTTP_USER_AGENT'])){
echo "Votre navigateur semble etre:\n";
echo getBrowser($_SERVER['HTTP_USER_AGENT']);
}

// Test avec des UserAgent connus:
$FF="Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:32.0) Gecko/
 ↪20100101 Firefox/32.0";
$msie="Mozilla/5.0 (compatible; MSIE 10.6; Windows NT 6.1; Trident/
 ↪5.0; InfoPath.2; SLCC1; .NET CLR 3.0.4506.2152; .NET CLR 3.5.
 ↪30729; .NET CLR 2.0.50727) 3gpp-gba UNTRUSTED/1.0";
$chrome="Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.
 ↪36 (KHTML, like Gecko) Chrome/37.0.2049.0 Safari/537.36";
$safari="Mozilla/5.0 (iPad; CPU OS 6_0 like Mac OS X) AppleWebKit/
 ↪536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A5355d Safari/
 ↪8536.25";
```

(suite sur la page suivante)

(suite de la page précédente)

```
echo "<br/> Test Firefox:<br/>\n";
echo getBrowser($FF)."<br/>\n";
echo "<br/> Test MSIE: \n";
echo getBrowser($msie)."<br/>\n";
echo "<br/> Test Chrome: \n";
echo getBrowser($chrome)."<br/>\n";
echo "<br/> Test Safari: \n";
echo getBrowser($safari);
?>
```

Résultat :

```
<br/> Test Firefox:<br/>
Firefox/32.0<br/>
<br/> Test MSIE:
MSIE 10.6<br/>
<br/> Test Chrome:
Chrome/37.<br/>
<br/> Test Safari:
Safari Version/6.0
```

3.11 Le typage en PHP

3.11.1 Les fonctions *gettype( )* et *settype( )* :

gettype( ) renvoie l'un des résultats suivants :

- integer
- double
- string
- array
- object
- class
- « unknown type »

settype( ) change le type d'un élément :

```
<?php
$a=3.5;
settype($a,"integer");
```

(suite sur la page suivante)

(suite de la page précédente)

```
echo "le contenu de la variable a est ".$a;
?>
```

dont le résultat sera :

```
le contenu de la variable a est 3
```

3.11.2 Fonctions de test :

- *is_int()*
- *is_long()*
- *is_double()*
- *is_array()*
- *is_object()*
- *is_string()*

Attention : N'oubliez pas comme en JavaScript la différence entre l'opérateur == et ===

Le premier vérifie l'égalité des contenus en ne tenant pas compte d'une éventuelle différence de typage (int ou string par exemple) tandis que le second vérifie une égalité stricte.

En d'autres termes : 5 == « 5 » est VRAI tandis que 5 === « 5 » est FAUX

3.12 Quelques particularités de PHP

3.12.1 Valeurs des variables :

```
    <?php
$toto = "Bonjour<br/>\n";
$var = "toto";
echo $$var;
    ?>
```

dont le résultat sera toujours :

3.12.2 Résultat brut

```
Bonjour<br/>
```

3.12.3 La fonction *eval()* :

Permet l'évaluation d'expressions arithmetiques directement en PHP. Existe aussi en JavaScript. Délicat à manipuler, problématique en termes de sécurité.

3.13 Les tableaux en PHP

3.13.1 Tableaux associatifs - parcours avec boucle foreach :

```
<?php
  $jours=array("Lu"=>"Lundi","Ma"=>"Mardi",
        "Me"=>"Mercredi","Je"=>"Jeudi","Ve"=>"Vendredi",
        "Sa"=>"Samedi","Di"=>"Dimanche" );

  foreach($jours as $key=>$val) echo $key." ".$val."<br>\n";
?>
```

Ce qui donne :

```
Lu Lundi
Ma Mardi
Me Mercredi
Je Jeudi
Ve Vendredi
Sa Samedi
Di Dimanche
```

3.13.2 Affichage avec print_r() :

```
<?php
     print_r($jours);
?>
```

3.13.3 Résultat brut html :

```
Array
(
    [Lu] => Lundi
    [Ma] => Mardi
    [Me] => Mercredi
    [Je] => Jeudi
    [Ve] => Vendredi
```

(suite sur la page suivante)

(suite de la page précédente)

```
    [Sa] => Samedi
    [Di] => Dimanche
)
```

3.13.4 Essayez vous-même

tabs

3.13.5 Utilisation de la fonction array_walk :

```
<?php array_walk($jours,'aff_tab'); ?>
```

En ayant défini au préalable :

```
<?php
function aff_tab($val, $key){
echo "$key-$val<br/>\n";
}
?>
```

On obtient le même résultat qu'avec la boucle foreach

3.13.6 Tri simple d'un tableau :

```
<?php
  sort($jours);
  array_walk($jours,'aff_tab');
?>
```

3.13.7 On obtient :

```
0-Dimanche
1-Jeudi
2-Lundi
3-Mardi
4-Mercredi
5-Samedi
6-Vendredi
```

C'est à dire que :

— Le tableau est trié selon l'ordre de ses valeurs

— les clefs sont effacées et réaffectées avec des entiers.

Si on veut préserver également les clefs du tableau associatif, il faut utiliser la méthode suivante :

3.13.8 Tri selon l'ordre naturel avec natsort

```php
<?php
    $jours=array("Lu"=>"Lundi","Ma"=>"Mardi",
    "Me"=>"Mercredi","Je"=>"Jeudi","Ve"=>"Vendredi",
    "Sa"=>"Samedi","Di"=>"Dimanche" );
    var_dump($jours);
    natsort($jours);
    var_dump($jours);
?>
```

3.13.9 Résultat brut html

```
array(7) {
  ["Lu"]=>
  string(5) "Lundi"
  ["Ma"]=>
  string(5) "Mardi"
  ["Me"]=>
  string(8) "Mercredi"
  ["Je"]=>
  string(5) "Jeudi"
  ["Ve"]=>
  string(8) "Vendredi"
  ["Sa"]=>
  string(6) "Samedi"
  ["Di"]=>
  string(8) "Dimanche"
}
array(7) {
  ["Di"]=>
  string(8) "Dimanche"
  ["Je"]=>
  string(5) "Jeudi"
  ["Lu"]=>
  string(5) "Lundi"
  ["Ma"]=>
  string(5) "Mardi"
  ["Me"]=>
  string(8) "Mercredi"
  ["Sa"]=>
  string(6) "Samedi"
```

(suite sur la page suivante)

(suite de la page précédente)

```
  ["Ve"]=>
  string(8) "Vendredi"
}
```

3.13.10 Exécution

tritabnat

On peut aussi utiliser la fonction natcasesort() si on ne veut pas se préoccuper de la casse des chaines présentes dans le tableau, soit à peu près l'ordre du dictionnaire . . .

3.14 Les tableaux prédéfinis de PHP

3.14.1 Les tableaux concernant le protocole HTTP :

— $_GET[ ], $_POST[ ] ou $_REQUEST[ ] qui englobe les 2
— $_SERVER[ ] : Variables décrivant le client ou la page courante
— $_GLOBALS[ ] variables globales
— $_COOKIE[ ] pour les cookies
— $_SESSION[ ] pour les sessions

3.14.2 Exemple récupération de $_SERVER[ ] grâce à la fonction getenv() :

```
<?php
function infos(){
  $env = array('remote_addr','http_accept_language','http_
↪host',
  'http_user_agent','script_filename','server_addr',
  'server_name','server_signature','server_software',
  'request_method','query_string','request_uri','script_name
↪');

  // Construction d'un tableau associatif
  // Avec les valeurs lues dans l'environnement
  $retour =array();
  foreach ($env as $clef) $retour[$clef] = getenv($clef);
  return $retour;
}

echo("Voici les infos disponibles:<BR>");
$tab = infos();
```

(suite sur la page suivante)

(suite de la page précédente)

```
foreach ($tab as $clef=>$val) echo $clef." :".$val."<br>\n";
?>
```

3.14.3 Résultat

```
Voici les infos disponibles:
remote_addr :::1
http_accept_language :fr-fr
http_host :localhost
http_user_agent :Mozilla/5.0 (Macintosh; U; Intel Mac OS X␣
↪10_6_4; fr-fr)
AppleWebKit/533.18.1 (KHTML, like Gecko) Version/5.0.2␣
↪Safari/533.18.5
script_filename :/Users/roza/Sites/php/exemples/infospy.php
server_addr :::1
server_name :localhost
server_signature :
server_software :Apache/2.2.14 (Unix) mod_ssl/2.2.14
OpenSSL/0.9.8l DAV/2 PHP/5.3.2
request_method :GET
query_string :
request_uri :/~roza/php/exemples/infospy.php
script_name :/~roza/php/exemples/infospy.php
`User-Agent <http://localhost/~roza/php/exemples/infospy.
↪php>`_
```

3.14.4 Exécution

infospy

3.15 L'inclusion de fichiers externes

3.15.1 *include* :

— Semblable aux *include* du C/C++
— Réalise une inclusion physique du fichier demandé

3.15.2 *include_once* :

— identique au include
— protège contre d'éventuelles inclusions multiples

— qui pourraient mener à des erreurs (redéclarations, etc.)

```
<?php include_once("connect.php"); ?>
```

3.15.3 *require* et *require_once* :

— fonctionnent comme le include et le include_once respectivement
— mais le programme s'arrête si le fichier inclus n'existe pas

```
<?php
    require("malib.php");
    require_once("connect.php");
?>
```

3.15.4 *dirname()*

Pour savoir dans quel répertoire on se trouve on peut utiliser la fonction PHP dirname()

```
<?php
include_once(dirname(__FILE__) . '/config/config.inc.php');
?>
```

Indication : Lorsqu'on inclus ou désigne des fichiers, il vaut mieux utiliser des chemins relatifs pour repérer les fichiers (comme ci dessus) plutôt que de donner un chemin absolu par rapport à la racine du serveur du style /home/user/www/config/config.inc.php Cela sera beaucoup plus portable d'un serveur à l'autre et vous évitera bien des déboires !

Avertissement : L'utilisation systématique de la version avec *once* (include_once ou require_once)n'est pas recommandée car elle peut causer des ralentissements à l'exécution du programme.

3.16 Les Objets en PHP

3.16.1 Evolutions et grands principes :

— Les objets existent en PHP à partir de la version 4
— Changements importants en PHP 5 : Convergence vers le modèle objet de Java
— Introduction comme en Java d'interfaces et de classes abstraites
— emploi des modifieurs *private* et *public* comme en java
— On retrouve aussi **__toString()**, **__clone()** et un mécanisme de traitement des exceptions semblable à celui de Java.

— Les constructeurs s'appellent désormais : **__construct()**
— et les destructeurs **__destruct()**
— les méthodes portent le mot clef *function* mais ne signalent pas leur type de retour
— les commentaires de documentation se font à la manière de Java

---

Indication : Les objets s'instancient avec le mot clef **new** comme en Java ou C++ mais on utilise **->** pour signifier l'appel d'une méthode. Le **.** est déjà pris pour la concaténation des chaînes...

3.16.2 Un Objet Simple Etudiant en PHP

Fabriquons maintenant un objet simple en PHP. Ecrivons un objet représentant un étudiant avec ses données :
— identifiant
— nom
— date de naissance
et des méthodes pour opérer sur ces données :
— constructeur
— getters et setters
— equals()
— toString() pour affichage

ce qui donne le code suivant :

```
<?php
/** Classe Etudiant en PHP */

class Etudiant{
  /** Identification unique d'un etudiant */
  protected $etudiant_id;
  /** Nom de l'etudiant */
  protected $nom;
  /** Date de naissance de l'etudiant */
  protected $naissance;

  public function __construct($id, $nom,$naissance){
    $this->etudiant_id = (int)$id;  // cast vers integer
    $this->nom = (string)$nom;      // cast vers string
    $this->naissance= (int)$naissance;  // cast vers date(timestamp)
  }

  /**
   * Fonction de comparaison simplifiee entre etudiants
   * == comparera id, nom et naissance
   */
```

(suite sur la page suivante)

(suite de la page précédente)

```
  public function equals(etudiant $etudiant){
    return ($this->getId() == $etudiant->getId());
  }

  public function getId(){
    return $this->etudiant_id;
  }

  public function getNom(){
    return $this->nom;
  }

  public function getNaissance(){
    return $this->naissance;
  }

  public function __toString(){
          setlocale(LC_TIME, "fr_FR");
          $ne=strftime('%A %d %B %Y',$this->naissance);
    return 'etudiant: id=' . $this->getId().', nom='.$this->
↪getNom()." $ne";
  }
}
/* Test : */
date_default_timezone_set('Europe/Paris');
$etu=new Etudiant(234,"Talon",time());
var_dump($etu);
echo "<br/>";
echo $etu;
?>
```

Resultat :

```
object(Etudiant)#1 (3) {
  ["etudiant_id":protected]=>
  int(234)
  ["nom":protected]=>
  string(5) "Talon"
  ["naissance":protected]=>
  int(1537386847)
}
<br/>etudiant: id=234, nom=Talon Mercredi 19 septembre 2018
```

Execution

etudiant php

Indication : Lorsqu'on utilise l'opérateur de comparaison **==** sur des objets, ceux-ci sont comparés en utilisant les règles suivantes :

deux objets sont égaux s'ils sont **instances de la même classe et ont les mêmes attributs et valeurs**, les valeurs étant comparées avec l'opérateur ==.

Lors de l'utilisation de l'opérateur d'identité **===**, les objets sont identiques **uniquement s'ils font référence à la même instance de la même classe**.

Avertissement : Beaucoup de programmeurs débutants se contentent d'utiliser PHP comme un langage purement procédural sans utiliser son côté objet. Ceci les bloquera tôt ou tard dans l'apprentissage du langage et des grands frameworks PHP. Si vous êtes trop mal à l'aise avec les Objets, suivez d'abord un cours d'introduction à la programmation Objet, en Java, Python ou C++ par exemple.

3.17 Les collections en PHP

3.17.1 En PHP standard, Collections = Arrays :

— Si on se contente de ce qu'offre PHP en standard, les collections se résument à l'utilisation des tableaux associatifs en PHP
— Le framework des *Collections* en Java est beaucoup plus riche

3.17.2 DataStructures ds :

Il faut installer une librairie supplémentaire *ds* (DataStructures) pour avoir accès à un Framework similaire en PHP.

Voir Documentation Collections php (http ://php.net/manual/fr/book.ds.php)

On va pour utiliser la commande *pecl* pour installer *ds* comme une extension :

```
pecl install ds
```

Puis charger l'extension en ajoutant aux fichiers .ini de PHP :

```
extension=ds.so
```

Cette extension nous donne accès à des classes similaires à celles du framework des Collections en Java. Les classes proposées sont par exemple :

— Sequence
— Vector
— Deque
— Vector

— Pair
— Set
— Stack
— Queue
— PriorityQueue

3.17.3 Utilisation de Ds :

Les classes et interfaces de ds s'utilisent dans un espace de nommage **Ds** :

```
<?php
$vector = new \Ds\Vector();
$vector->push("un");
$vector->push("deux");
$vector->push("trois", "quatre");
// ...[ , ] = unpacking
$vector->push(...["cinq", "six"]);
print_r($vector);
?>
```

3.17.4 Exemple d'utilisation de la classe *Set*

Utilisons maintenant concrètement la classe *Set*.

Question HTML dans un select multiple :

Prenons un petit formulaire en HTML qui propose un choix de couleurs dans un select :

```
<p>Quelles sont les couleurs du maillot des panthères du Fleury␣
↪Loiret Handball ?
  <select name="coul[]" multiple size=5>
      <option value="jaune">Jaune
      <option value="rose">Rose
      <option value="bleu">Bleu
      <option value="noir">Noir
      <option value="blanc">Blanc
      <option value="vert">Vert
  </select>
</p>
```

Réponse PHP avec Collections

```
<?php
if (!empty($_GET['coul'])){
```

(suite sur la page suivante)

(suite de la page précédente)

```
    $couleursReponse = new \Ds\Set($_GET['coul']);
    $couleursCorrectes = new \Ds\Set(['rose','noir','blanc
↪']);
// Calculons la différence des 2 ensembles et voyons si␣
↪elle est vide
if (($couleursReponse->diff($couleursCorrectes))->isEmpty())
        echo "Bravo les couleurs de Fleury Hand Ball sont␣
↪bien Rose Noir et Blanc !";
else
        echo "Mauvaise réponse : les couleurs de Fleury␣
↪Hand Ball sont: Rose Noir et Blanc !";
}
```

Sans Collections, on utilise les tableaux . . .

Sans **Ds**, nous aurions été obligés de nous contenter de tableaux PHP et d'utiliser par exemple la méthode **array_diff** : Voir : array_diff php (http ://php.net/manual/fr/function.array-diff.php)

Avec des Arrays :

On peut ici s'en sortir avec de simples tableaux PHP en vérifiant que la taille du tableau des réponses données est la même que celle du tableau des bonnes réponses, puis que les contenus de ces tableaux sont identiques.

```
<?php
if (!empty($_GET['coul'])){
    $couleursReponse = $_GET['coul'];
    $couleursCorrectes = array(['rose','noir','blanc']);
    if (count($couleursReponse) == count($couleursCorrectes) && !
↪array_diff($couleursReponse, $couleursCorrectes))
        echo "Bravo les couleurs de Fleury Hand Ball sont bien Rose␣
↪Noir et Blanc !";
else
        echo "Mauvaise réponse : les couleurs de Fleury Hand Ball␣
↪sont: Rose Noir et Blanc !";
}
```

3.18 Connexion aux bases de données depuis PHP

3.18.1 Une table simple en SQL :

```
CREATE TABLE `CARNET` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `NOM` varchar(30) DEFAULT NULL,
  `PRENOM` varchar(30) DEFAULT NULL,
  `NAISSANCE` date DEFAULT NULL,
  `VILLE` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8;

INSERT INTO `CARNET` VALUES
(1,'SMITH','JOHN','1980-12-17','ORLEANS'),
(2,'DURAND','JEAN','1983-01-13','ORLEANS'),
(3,'GUDULE','JEANNE','1967-11-06','TOURS'),
(4,'ZAPATA','EMILIO','1956-12-01','ORLEANS'),
(5,'JOURDAIN','NICOLAS','2000-09-10','TOURS'),
(6,'DUPUY','MARIE','1986-01-11','BLOIS'),
(7,'ANDREAS','LOU','1861-02-12','ST Petersbourg'),
(9,'Kafka','Franz','1883-07-03','Prague'),
(11,'Dalton','Joe','2003-12-06','Dallas');
```

On insère cette table dans MySQL en ligne de commande ou à l'aide de PHPMyAdmin. Puis, pour consulter cette table depuis PHP, on utilise le connecteur PDO qui offre une interface de connexion utilisable pour tous les SGBD (Systemes de Gestion de Bases de Donnees) habituels comme MySQL, PostgreSQL, Oracle ou Microsoft SQL Server.

3.18.2 Connexion Simple en PHP avec PDO

```
<!doctype html>
<html>
<head>
<title>
Connexion à MySQL avec PDO
</title>
<meta charset="utf-8">
</head>
<body>
<h1>
Interrogation de la table CARNET avec PDO
</h1>

<?php
require("connect.php");
// pour oracle: $dsn="oci:dbname=//serveur:1521/base
// pour sqlite: $dsn="sqlite:/tmp/base.sqlite"
$dsn="mysql:dbname=".BASE.";host=".SERVER;
    try{
      $connexion=new PDO($dsn,USER,PASSWD);
    }
```

(suite sur la page suivante)

(suite de la page précédente)

```
    catch(PDOException $e){
      printf("Échec de la connexion : %s\n", $e->getMessage());
      exit();
    }
$sql="SELECT * from CARNET";
if(!$connexion->query($sql)) echo "Pb d'accès au CARNET";
else{
     foreach ($connexion->query($sql) as $row)
     echo $row['PRENOM']." ".
          $row['NOM']."né(e) le ".
          $row['NAISSANCE']."<br/>\n";
}
?>
</body>
</html>
```

Avec un fichier connect.php contenant les informations de connexion au serveur MySQL :

```
<?php
define('USER',"scott");
define('PASSWD',"tiger");
define('SERVER',"localhost");
define('BASE',"dbscott");
?>
```

Resultat brut html :

```
<!doctype html>
<html>
<head>
<title>
Connexion à MySQL avec PDO
</title>
<meta charset="utf-8">
</head>
<body>
<h1>
Interrogation de la table CARNET avec PDO
</h1>


JOHN SMITHné(e) le 1980-12-17<br/>
JEAN DURANDné(e) le 1983-01-13<br/>
JEANNE GUDULEné(e) le 1967-11-06<br/>
EMILIO ZAPATAné(e) le 1956-12-01<br/>
NICOLAS JOURDAINné(e) le 2000-09-10<br/>
MARIE DUPUYné(e) le 1986-01-11<br/>
```

(suite sur la page suivante)

(suite de la page précédente)

```
LOU ANDREASné(e) le 1861-02-12<br/>
Franz Kafkané(e) le 1883-07-03<br/>
Joe Daltonné(e) le 2003-12-06<br/>
</body>
</html>
```

Execution

carnet php

Fabrication d'une table HTML avec les résultats :

```
<html>
<head>
<title>
Connexion à MySQL avec PDO
</title>
<meta charset="utf-8">
<body>
<h1>
Interrogation de la table CARNET avec PDO
</h1>

<?php
require("connect.php");
$dsn="mysql:dbname=".BASE.";host=".SERVER;
    try{
      $connexion=new PDO($dsn,USER,PASSWD);
    }
    catch(PDOException $e){
      printf("Échec de la connexion : %s\n", $e->getMessage());
      exit();
    }
$sql="SELECT * from CARNET";
if(!$connexion->query($sql)) echo "Pb d'accès au CARNET";
else{
?>
<table>
<tr> <td> Nom </td> <td> Prénom </td></tr>
  <?php
  foreach ($connexion->query($sql) as $row)
echo "<tr><td>".$row['NOM']."</td><td>".
  $row['PRENOM']."</td></tr>\n";
  ?>
</table>
<?php } ?>
```

(suite sur la page suivante)

(suite de la page précédente)

```
</body>
</html>
```

Résultats bruts :

```
<html>
<head>
<title>
Connexion à MySQL avec PDO
</title>
<meta charset="utf-8">
<body>
<h1>
Interrogation de la table CARNET avec PDO
</h1>


<table>
<tr> <td> Nom </td> <td> Prénom </td></tr>
  <tr><td>SMITH</td><td>JOHN</td></tr>
<tr><td>DURAND</td><td>JEAN</td></tr>
<tr><td>GUDULE</td><td>JEANNE</td></tr>
<tr><td>ZAPATA</td><td>EMILIO</td></tr>
<tr><td>JOURDAIN</td><td>NICOLAS</td></tr>
<tr><td>DUPUY</td><td>MARIE</td></tr>
<tr><td>ANDREAS</td><td>LOU</td></tr>
<tr><td>Kafka</td><td>Franz</td></tr>
<tr><td>Dalton</td><td>Joe</td></tr>
</table>
</body>
</html>
```

Execution

Carnet Table

On peut faire un petit refactoring avec une fonction qui établit la connexion à la base :

```
<?php

require("connect.php");
function connect_bd(){
        $dsn="mysql:dbname=".BASE.";host=".SERVER;
                try{
                $connexion=new PDO($dsn,USER,PASSWD);
                }
```

(suite sur la page suivante)

(suite de la page précédente)

```
                catch(PDOException $e){
                printf("Échec de la connexion : %s\n", $e->
↪getMessage());
                exit();
                }
        return $connexion;
}

?>
```

et améliorer l'affichage des résultats :

```
<html>
<head>
<title>
Connexion à MySQL avec PDO
</title>
<meta charset="utf-8">
<link rel="stylesheet" href="tabstyle.css" />
</head>
<body>
<h1>
Interrogation de la table CARNET avec PDO
</h1>

<?php
require_once('connexion.php');
$connexion=connect_bd();
$sql="SELECT * from CARNET";
if(!$connexion->query($sql)) echo "Pb d'accès au CARNET";
else{
?>
<table class="centre" id="jolie">
<tr> <td> ID </td> <td> Prénom </td> <td> Nom </td><td> Naissance </
↪td> </tr>
  <?php
  foreach ($connexion->query($sql) as $row)
echo "<tr><td>".$row['ID']."</td>
          <td>".$row['PRENOM']."</td>
          <td>".$row['NOM']."</td>
          <td>".$row['NAISSANCE']."</td></tr><br/>\n";
  ?>
</table>
<?php } ?>
</body>
</html>
```

Avec le fichier CSS :

```
/* Bordure simple autour des tableaux */
table,th, td { border: 1px solid grey;}
table{border-collapse:collapse;}
/* Centrage tableau */
table.centre{ margin:auto;}
/* centrage du texte dans les cellules du tableau */
table.centre td{text-align:center;}

table#jolie tr:first-child{
        background:LightPink;
}
table#jolie tr:nth-child(2n){
        background:#EFD3C9;
}
table#jolie tr:nth-child(2n+3){
        background:#BCBCD0;
}
/* si un tableau a une seule ligne on l'affiche en rouge */
table tr:only-child{
        background:red;
}
```

Résultats bruts :

```
<html>
<head>
<title>
Connexion à MySQL avec PDO
</title>
<meta charset="utf-8">
<link rel="stylesheet" href="tabstyle.css" />
</head>
<body>
<h1>
Interrogation de la table CARNET avec PDO
</h1>


<table class="centre" id="jolie">
<tr> <td> ID </td> <td> Prénom </td> <td> Nom </td><td> Naissance </
↪td> </tr>
  <tr><td>1</td>
          <td>JOHN</td>
          <td>SMITH</td>
          <td>1980-12-17</td></tr><br/>
<tr><td>2</td>
```

(suite sur la page suivante)

(suite de la page précédente)

```
          <td>JEAN</td>
          <td>DURAND</td>
          <td>1983-01-13</td></tr><br/>
<tr><td>3</td>
          <td>JEANNE</td>
          <td>GUDULE</td>
          <td>1967-11-06</td></tr><br/>
<tr><td>4</td>
          <td>EMILIO</td>
          <td>ZAPATA</td>
          <td>1956-12-01</td></tr><br/>
<tr><td>5</td>
          <td>NICOLAS</td>
          <td>JOURDAIN</td>
          <td>2000-09-10</td></tr><br/>
<tr><td>6</td>
          <td>MARIE</td>
          <td>DUPUY</td>
          <td>1986-01-11</td></tr><br/>
<tr><td>7</td>
          <td>LOU</td>
          <td>ANDREAS</td>
          <td>1861-02-12</td></tr><br/>
<tr><td>9</td>
          <td>Franz</td>
          <td>Kafka</td>
          <td>1883-07-03</td></tr><br/>
<tr><td>11</td>
          <td>Joe</td>
          <td>Dalton</td>
          <td>2003-12-06</td></tr><br/>
</table>
</body>
</html>
```

Execution

Carnet Table Version2

On peut générer des pages différentes avec des listes déroulantes ou des listes de liens, listes à puces etc.

Création d'une liste déroulante :

```
<!doctype html>
<html>
```

(suite sur la page suivante)

(suite de la page précédente)

```
<head>
<title>
Connexion à MySQL avec PDO
</title>
<meta charset="utf-8">
</head>
<body>
<h1>
Interrogation de la table CARNET avec PDO
</h1>
<?php
require_once('connexion.php');
$connexion=connect_bd();
$sql="SELECT * from CARNET";
if(!$connexion->query($sql)) echo "Pb d'accès au CARNET";
else {
      ?>
      <form action="recherche.php" method="GET">
             <select name="ID">
       <?php
       foreach ($connexion->query($sql) as $row)
       if(!empty($row['NOM']))
             echo "<option value='".$row['ID']."'>"
         .$row['PRENOM']." ".$row['NOM']."</option>\n";
       ?>
      </select>
      <input type="submit" value="Rechercher">
      </form>
      <?php
       }
      ?>
</body>
</html>
```

Remarquez l'usage de la clef primaire de la table comme value des options, ce qui assurent l'unicité des valeurs et évite toute amiguité.

Résultats bruts :

```
<!doctype html>
<html>
<head>
<title>
Connexion à MySQL avec PDO
</title>
<meta charset="utf-8">
</head>
<body>
```

(suite sur la page suivante)

(suite de la page précédente)

```
<h1>
Interrogation de la table CARNET avec PDO
</h1>

     <form action="recherche.php" method="GET">
            <select name="ID">
      <option value='1'>JOHN SMITH</option>
<option value='2'>JEAN DURAND</option>
<option value='3'>JEANNE GUDULE</option>
<option value='4'>EMILIO ZAPATA</option>
<option value='5'>NICOLAS JOURDAIN</option>
<option value='6'>MARIE DUPUY</option>
<option value='7'>LOU ANDREAS</option>
<option value='9'>Franz Kafka</option>
<option value='11'>Joe Dalton</option>
     </select>
     <input type="submit" value="Rechercher">
     </form>
     </body>
</html>
```

Execution

Carnet Select

3.19 Requêtes préparées en PHP

3.19.1 Recherche simple en PHP avec PDO

```
<!doctype html>
<html>
<head>
<title>
Recherche d'une personne par ID
</title>
<meta charset="utf-8">
</head>
<body>
<?php $wanted=$_GET['ID'];
if (!empty($wanted)){
        echo "<h1>Recherche de  $wanted </h1>";
        require_once('connexion.php');
        $connexion=connect_bd();
        $sql="SELECT * from CARNET where ID='".$wanted."'";
```

(suite sur la page suivante)

(suite de la page précédente)

```
        if(!$connexion->query($sql))
                echo "Pb de requete";
        else{
        foreach ($connexion->query($sql) as $row)
                echo $row['NOM']." ".$row['PRENOM']."</br>\n";
  }
}
  ?>
</body>
</html>
```

Appel avec le paramètre ID passé sur l'URL : php exemples/pdo/recherche.php ?ID=3

Execution

recherche php

Mais lorsqu'il y a de nombreux paramètres, on se retrouve avec de nombreuses concaténations de chaines entourées de "cotes" ce qui est une grande source d'erreurs et de lenteurs d'écriture. Pour remédier à cela, on peut utiliser des requêtes préparées qui permettent de bien dissocier la requête des paramètres qui vont lui être fournis avant son exécution. Ces *PreparedStatement* seront également bien préférables en termes de sécurité et à utiliser **systématiquement**.

3.19.2 Recherche avec PreparedStatement

```
<!doctype html>
<html>
<head>
<title>
Recherche d'une personne par ID
</title>
<meta charset="utf-8">
</head>
<body>
<?php $wanted=$_GET['ID'];
if (!empty($wanted)){
  echo "<h1>Recherche de  $wanted </h1>";
  require_once('connexion.php');
  $connexion=connect_bd();
$sql="SELECT * from CARNET where ID=:id";
$stmt=$connexion->prepare($sql);
$stmt->bindParam(':id', $_GET['ID']);
$stmt->execute();
if (!$stmt) echo "Pb d'accès au CARNET";
else{
```

(suite sur la page suivante)

(suite de la page précédente)

```
    if ($stmt->rowCount()==0) echo "Inconnu !<br/>";
    else
     foreach ($stmt as $row)
     echo $row['PRENOM']." ".$row['NOM'].
          "né(e) le ".$row['NAISSANCE']."<br/>";
}
}
?>
</body>
</html>
```

Les requêtes préparées limitent fortement la possibilité d'*injections SQL* comme nous le verront plus tard.

3.20 Compléments sur PDO - Sécurité

3.20.1 Filtrage d'entrées

On peut vouloir nourrir une requête directement avec des données provenant d'un formulaire :

```
<?php
  $sql = sprintf(
    'SELECT id FROM CARNET WHERE email = "%s"',$_GET['email'])
  );
?>
```

On voit bien que la valeur de l'entrée email dans le tableau _GET n'est absolument pas vérfifiée avant son utilisation !

On peut essayer dans ce cas d'utiliser un filtre PHP pour contrôler un peu les choses :

```
<?php
  $sql = sprintf(
       'SELECT id FROM CARNET WHERE email = "%s"',
       filter_input(INPUT_GET, 'email')
  );
?>
```

Mais ceci constitue une manière peu sûre de procéder malgré l'utilisation du filtre PHP. Cela laisse en effet la possiblilité d'insertion de code malveillant non contrôlé.

L'exemple classique est la requête SQL construite dans la chaine suivante :

```
<?php
  $sql = "SELECT nom FROM USERS WHERE login='".
         $_REQUEST['login']."' AND PASSWD='".
         $_REQUEST['pass']."'";
?>
```

Qui donne lors de son exécution avec $_REQUEST[“login”] = » toto” – » :

```
SELECT nom FROM USERS WHERE login='toto'  -- ' AND PASSWD= '".$_
↪REQUEST['pass']."' ";
```

Avertissement : Le - - constituant un début de commentaire SQL, ceci constitue une injection SQL qui est l'une des principales failles de sécurité exploitées par les Hackers. Pour s'en prémunir, il faut utiliser **à la fois** le filtrage des entrées et les requêtes préparées.

```
<?php
$sql = 'SELECT id FROM CARNET WHERE email = :email';
$statement = $pdo->prepare($sql);
$email = filter_input(INPUT_GET, 'email');
$statement->bindValue(':email', $email);
?>
```

Il faut parfois préciser dans un troisième argument le type des paramètres attendus :

```
<?php
  $sql = 'SELECT email FROM CARNET WHERE id = :id';
  $statement = $pdo->prepare($sql);
  $userId = filter_input(INPUT_GET, 'id');
  $statement->bindValue(':id', $userId, PDO::PARAM_INT);
?>
```

3.21 Filtrage en PHP

Les vérifications dans les formulaires HTML5 et en JavaScript sont valables uniquement côté client. Pour des raisons de sécurité, il faut réeffectuer complètement toutes les vérifications côté serveur. PHP met pour cela à la disposition des programmeurs tout un ensemble de filtres. La première des vérifications consiste à bien prendre en compte les caractères spéciaux.

3.21.1 Gestion des caractères spéciaux :

Les « magic quotes » :

Avant PHP 5.4, une directive concernant ces magic quotes existait. Si dans le fichier de configuration de PHP, la directive magic_quotes_gpc etait activée, PHP modifiait automatiquement certaines entrées de formulaires en procédant à des protections de certains caractères spéciaux par des insertion de « backslashes ». Par exemple

— les caractères accentués
— les apostrophes
— les backslashes

Un peu à la manière de la fonction addslashes().

Cette protection était destinée à préparer les données avant des requêtes SQL pour empêcher une éventuelle injection SQL. Ce comportement automatique est toutefois parfois gênant si on veut simplement réafficher les chaines saisies et non pas les utiliser dans des requêtes SQL. En outre, on ne veut pas toujours protéger les chaines de la même façon selon l'usage qu'on veut en faire par la suite.

On peut vouloir dans certains cas, protéger des chaines par exemple :

— htmlspecialchars() pour éviter des injections de code HTML
— PDO : :quote() pour se protéger d'injections SQL

```
<?php
  $pdo = new PDO('sqlite:./tmp/mydb.sqlite');
  $string = 'Orléans';
  print "Chaîne sans quotes: $string\n";
  print "Chaîne avec quotes: " . $pdo->quote($string) . "\n";
```

Cela ne vous dispense pas d'utiliser les *PreparedStatement* vus précédemment.

Les filtres PHP :

Les plus directs à utiliser sur les formulaires sont basés sur la fonction filter_input() avec en paramètre INPUT_GET ou INPUT_POST Voici quelques exemples typiques :

```
<?php
    $entier = filter_input(INPUT_POST, 'var1', FILTER_SANITIZE_
↪NUMBER_INT);

    $url = filter_input(INPUT_POST, 'var2', FILTER_VALIDATE_URL);

    if (!filter_input(INPUT_GET, "email",  FILTER_VALIDATE_EMAIL))
        echo("Email non valide");
    else echo("Email valide");
    ?>
```

On peut aussi filtrer des carctères spéciaux :

```
<?php
$search_html = filter_input(INPUT_GET, 'search', FILTER_SANITIZE_
↪SPECIAL_CHARS);
$search_url = filter_input(INPUT_GET, 'search', FILTER_SANITIZE_
↪ENCODED);
echo "Vous avez recherché $search_html.\n";
echo "<a href='?search=$search_url'>Nouvelle recherche.</a>";
?>
```

3.22 Gestion des transactions avec PDO

3.22.1 Problème

Une série de requêtes SQL sont logiquement liées entre elles et on voudrait qu'elles soient toutes exécutées ou aucune. En effet dans certains cas, la prise en compte d'une partie des requêtes seulement peut conduire à une incohérence dans le système d'information. La base de données peut ainsi être corrompue et très difficile à rectifier par la suite. Par exemple, si on a 2 requêtes qui se suivent et qui sont liées :

```
<?php
  require 'connexion.php';
  $connexion=connect_bd();

  $stmt1 = $pdo->prepare('
      UPDATE compte
      SET solde = solde - :montant
      WHERE nom = :nom
  ');
  $stmt2 = $pdo->prepare('
      UPDATE compte
      SET solde = solde + :montant
      WHERE nom = :nom
  ');

  // Retrait du Compte1
  $cpte1 = 'Compte1';
  $montant = 50;
  $stmt1->bindParam(':nom', $cpte1);
  $stmt1->bindParam(':solde', $montant, PDO::PARAM_INT);
  $stmt1->execute();

  // Credit du Compte2
  $cpte2 = 'Compte2';
  $depot = 50;
  $stmt2->bindParam(':nom', $cpte2);
  $stmt->bindParam(':montant', $depot, PDO::PARAM_INT);
  $stmt->execute();
?>
```

Ceci peut conduire à un problème en cas d'interruption de cette séquence. En particulier le Compte1 peut avoir été débité sans que le Compte2 soit crédité.

On peut résoudre cette fragilité en utilisant une transaction :

```
<?php
  require 'connexion.php';
  $connexion=connect_bd();
```

(suite sur la page suivante)

(suite de la page précédente)

```
  $stmt1 = $connexion->prepare('
      UPDATE compte
      SET solde = solde - :solde
      WHERE nom = :nom
  ');
  $stmt2 = $connexion->prepare('
      UPDATE compte
      SET solde = solde + :montant
      WHERE nom = :nom
  ');

  // On commence la transaction
  $connexion ->beginTransaction()
  // Retrait du Compte1
  $cpte1 = 'Compte1';
  $montant = 100;
  $stmt1->bindParam(':nom', $cpte1);
  $stmt1->bindParam(':solde', $montant, PDO::PARAM_INT);
  $stmt1->execute();

  // Credit du Compte2
  $cpte2 = 'Compte2';
  $depot = 50;
  $stmt2->bindParam(':nom', $cpte2);
  $stmt->bindParam(':montant', $depot, PDO::PARAM_INT);
  $stmt->execute();
  //on termine la transaction
  $connexion -> commit();
?>
```

Si la séquence échoue, PDO commandera un *RollBack* automatique, c'est à dire une annulation de toute la transaction.

3.23 Connexion persistante avec PDO

Pour créer une connexion persistante avec PDO, il suffit d'utiliser l'attribut *ATTR PERSISTENT* à l'instanciation de l'objet PDO. Lors des appels ultérieurs, si les paramètres de création sont identiques, l'objet déjà créé sera simplement réutilisé.

```
<?php
function connect_db()
{
  $dsn="mysql:dbname=".BASE.";host=".SERVER;
  try
  {
      $connexion=new PDO($dsn,USER,PASSWD,
          array(PDO::ATTR_PERSISTENT =>true));
```

(suite sur la page suivante)

(suite de la page précédente)

```
    }
    catch(PDOException $e)
    {
        printf("Échec de la connexion : %s\n", $e->getMessage());
        exit();
    }
    return $connexion;
}
?>
```

3.24 Validation et contrôle d'entrées avec GUMP

On peut également utiliser des librairies aidant à gérer la validation comme : GUMP (https ://github.com/Wixel/GUMP)

3.24.1 Pour installer :

Créons un fichier composer.json dans notre répertoire de travail :

```
{
    "require": {
        "wixel/gump": "dev-master"
     }
}
```

Puis utilisons l'outil PHP `composer` pour installer les composants demandés :

```
composer update
```

3.24.2 Nous pouvons utiliser Gump pour vérifier des entrées :

```
<?php
require "vendor/wixel/gump/gump.class.php";

$is_valid = GUMP::is_valid($_POST, array(
    'username' => 'required|alpha_numeric',
    'password' => 'required|max_len,100|min_len,6'
));

if($is_valid) {
        echo "valid username and password ";
} else {
    print_r($is_valid);
```

(suite sur la page suivante)

(suite de la page précédente)

```
}

?>
```

3.24.3 ou de manière plus détaillée :

```
<?php
require "vendor/wixel/gump/gump.class.php";

if ($_SERVER["REQUEST_METHOD"] == "POST"){
  $validator = new GUMP();

  $name = $_POST['name'];
  $password = $_POST['password'];

  $_POST = array(
    'name' => $name,
    'password' => $password);

  // néttoie les données et convertit les chaines en utf-8 si besoin
  $_POST = $validator->sanitize($_POST);

  // Définissons les règles et les filtres:
  $rules = array(
    'name' => 'required|alpha_numeric|max_len,100|min_len,6',
    'password' => 'required|max_len,100|min_len,6');

  $filters = array(
    'name'     => 'trim|sanitize_string',
    'password'     => 'trim|base64_encode');

  // On applique les filtres
  $_POST = $validator->filter($_POST, $filters);

  // On valide
  $is_valid  = $validator->validate($_POST, $rules);

  // On vérifie le résultat
  if ($is_valid === true )
  {
    echo $name;
    echo $password;
    exit;
  }
  else
  {
    echo "Erreurs détectées dans les entrées:\n";
```

(suite sur la page suivante)

(suite de la page précédente)

```
    // on affiche les erreurs en html
    echo $validator->get_readable_errors(true);
  }
}
```

3.25 Http et la conservation d'informations sur les clients

3.25.1 Problème

Le protocole HTTP est un protocole *sans mémoire*. Une requête d'un client ou les informations qui y sont rattachées ne sont pas mémorisées par défaut.

Plusieurs techniques ont été développées pour remédier à ce manque :

- Envoyer de l'information sur l'URL
- Utiliser un champ caché HTML
- Utiliser des Cookies
- Utiliser des Sessions

Envoi d'information sur l'URL :

Considérons une première page, page1.php :

```
<!doctype html>
<html lang="fr">
<head>
<meta charset="utf-8"/>
<title> Formulaires</title>
</head>
<body>
<?php
        if (!isset($_GET['login'])){
                ?>
        <form method='GET'
                  action=<?php echo $_SERVER['PHP_SELF'] ?>
        >
        <p>Login: <input type="text" name="login"></p>
        <input type="submit" value="Valider">
        </form>
        <?php
        }
        else {
        header('Location:page2.php?login='.$_GET['login']);
        }
?>
```

(suite sur la page suivante)

(suite de la page précédente)

```
</body>
</html>
```

qui se poursuit par une page2 :

```
<!doctype html>
<html lang="fr">
<head>
<meta charset="utf-8"/>
<title>Formulaires</title>
</head>
<body>
<?php
        if (isset($_GET['login'])){
                echo $_GET['login'];
        }
        else {
        header('Location:page1.php');
        }
?>
</body>
</html>
```

Exécution :

Passage d'info sur l'URL

Utiliser des cookies :

L'information est stockée dans un petit fichier texte sur le navigateur du client. On peut par exemple s'en servir pour stocker un identifiant de session, un login, un compteur de visites ou encore mesurer un temps de connexion.

```
<?php
if (isset($_COOKIE['compteur']))
  {
    $message = "Vous etes deja venu ".$_COOKIE['compteur']." fois
↪<br/>\n";
    $valeur = $_COOKIE['compteur'] + 1;
  }
  else
  {
     $message = "Je vous met un petit cookie<br/>\n";
     $valeur = 1;
  }
setCookie("compteur", $valeur);
```

(suite sur la page suivante)

(suite de la page précédente)

```
echo $message;
?>
```

Exécution :

Cookies en PHP

Mais si on a des informations en plus grand nombre à stocker ou qui revètent un caractère plus sensible, on préférera les stocker essentiellement côté serveur et utiliser le mécanisme plus complet des sessions. Celui-ci consiste à utiliser le tableau associatif _SESSION[] qui permet de stocker toute sorte d'informations associées à ce client (données de type nombres ou chaines, tableaux ou objets PHP).

Considérons une première page mettant en place une session :

```
<?php
// session1.php
session_start();
if (!isset($_SESSION['cpt']))
    $_SESSION['cpt']=0;
else
    $_SESSION['cpt']++;
echo "Vous avez vu cette page ".$_SESSION['cpt']." fois <br/>\n";
echo "Le SID courant est " . session_id();
echo "<br/> <a href=\"session2.php\">Aller à la page suivante␣
↪session2.php</a>";
?>
```

Puis on va relire les informations stockées en variables de session dans une autre page :

```
<?php
// session2.php
session_start();
if (!isset($_SESSION['cpt']))
    $_SESSION['cpt']=0;
else
    $_SESSION['cpt']++;
echo "bonjour {$_SESSION['login']} !<br>\n";
echo "vous avez vu cette page " . $_SESSION['cpt'] . " fois<br/>\n";
echo "Votre SID est toujours " . session_id();
echo "<br/> <a href=\"session1.php\">Retour a session1.php</a>";
?>
```

Exécution :

Utilisation variable de session PHP

Champs cachés

Un quatrième mécanisme est employé pour conserver de l'information dans des pages Web elles-mêmes comme l'utilisation de champs cachés : input de type *hidden*. Ces champs peuvent par exemple servir à stocker dans des formulaires HTML un champ spécial dit *csrf token* qui contiendra un identifiant unique temporaire pour se prémunir des attaques de type CSRF : Cross Site Request Forgery dont un exemple est l'envoi d'un mail contenant une image a quelqu'un et cette image est en fait un lien vers une page d'administration sur laquelle le destinataire du mail a des droits particuliers comme editer ou supprimer une ressource.

Cette page pourra déclencher une suppression ou une modification de contenu non souhaitée.

Les principaux Frameworks Web comme Symfony, Laravel en PHP ou Django, Flask en Python prennent en charge la génération automatique de ce token et sa mise en variable de session mais il faut tout de même l'appeler dans les formulaires ou lors de l'utilisation d'Ajax.

3.26 Manipuler XML avec PHP

Le format XML est utilisé de façon très variée. Nous le trouvons dans des services Web, des fichiers de configuration, des formats comme SVG, MathML, docx, odt, etc. Sa manipulation dans un langage comme PHP est donc particulièrement importante.

On peut manipuler XML de différentes manières

- A la main
- Avec XMLWriter/XMLReader
- Avec DOM
- Avec SimpleXML

ou des combinaisons de ces méthodes.

- DOM ou Document Object Model est une méthode qui recrée toute l'arborescence d'un document XML sous forme d'objets PHP. Son utilisation est simple mais elle est couteuse en ressources, en temps d'exécution et un peu verbeuse.
- XMLWriter et XMLReader traitent les fichiers XML a plus bas niveau mais leur utilisation exclusive rend parfois le code délicat à implémenter surtout pour la lecture de fichiers complexes.
- SimpleXML représente une sorte de compromis Simplicité/Performance.
- Traiter des fichiers xml « à la main » est généralement à éviter sauf pour créer des fichiers très simples.

3.26.1 Traitement de fichiers XML à la main :

Observons d'abord comment créer un fichier XML contenant une liste de programmes TV : La lecture de fichiers XML sans API est peu recommandée.

```
<?php
header('Content-Type: text/xml');
print '<?xml version="1.0"?>' . "\n";
print "<programmes>\n";
```

(suite sur la page suivante)

(suite de la page précédente)

```
$programmes = array(
    array('nom'=> 'Simpsons',
        'chaine'=> 'TF18',
        'debut' => '21:00',
        'duree' => '30'),
    array('nom'=> 'Blake et Mortimer',
        'chaine' => 'M54',
        'debut'=>'20:00','duree'=>'60'));
foreach ($programmes as $show) {
    print "\t<show>\n";
    foreach($show as $tag => $data) {
      print "\t<$tag>"
      . htmlspecialchars($data)
      . "\t</$tag>\n";
    }
    print "</show>\n";}
print "</programmes>\n";
?>
```

3.26.2 Exécution :

Ecriture XML à la main

3.26.3 Resultat brut html :

```
<?xml version="1.0"?>
<programmes>
        <show>
        <nom>Simpsons        </nom>
        <chaine>TF18        </chaine>
        <debut>21:00        </debut>
        <duree>30        </duree>
</show>
        <show>
        <nom>Blake et Mortimer        </nom>
        <chaine>M54        </chaine>
        <debut>20:00        </debut>
        <duree>60        </duree>
</show>
</programmes>
```

3.26.4 Ecriture avec XMLWriter :

Un exemple simple pour démarrer :

```php
<?php
$xml = new XMLWriter();
$xml->openURI('test.xml');
$xml->startElement('toto');
$xml->writeElement('url', 'http://totototo.com');
$xml->endElement();
$xml->flush();
?>
```

3.26.5 Resultat brut :

```
<toto><url>http://toto.com</url></toto>
```

et si on récupère des données de la table CARNET pour les exporter en XML :

```php
<?php
require_once('connexion.php');
$connexion=connect_bd();
$sql="SELECT * from CARNET";
$data=$connexion->query($sql);
$xml= new XMLWriter();
$xml->openUri("contacts.xml");
$xml->startDocument('1.0', 'utf-8');
$xml->startElement('mescontacts');

  while($pers=$data->fetch()){
    $xml->startElement('contact');
    $xml->writeAttribute('id', $pers['ID']);
    $xml->writeElement('prenom',$pers['PRENOM']);
    $xml->writeElement('nom',$pers['NOM']);
    $xml->writeElement('naissance',$pers['NAISSANCE']);
    $xml->endElement();
  }
$xml->endElement();
$xml->endElement();
$xml->flush();
  ?>
```

3.26.6 Resultat :

```
<?xml version="1.0" encoding="UTF-8"?>
<mescontacts>
    <contact id="1">
        <prenom>JOHN</prenom>
        <nom>SMITH</nom>
        <naissance>1980-12-17</naissance>
```

(suite sur la page suivante)

(suite de la page précédente)

```
    </contact>
    <contact id="2">
        <prenom>JEAN</prenom>
        <nom>DURAND</nom>
        <naissance>1983-01-13</naissance>
    </contact>
    <contact id="3">
        <prenom>JEANNE</prenom>
        <nom>GUDULE</nom>
        <naissance>1967-11-06</naissance>
        </contact>
    <contact id="4">
        <prenom>EMILIO</prenom>
        <nom>ZAPATA</nom>
        <naissance>1956-12-01</naissance>
        </contact>
    <contact id="5">
        <prenom>NICOLAS</prenom>
        <nom>JOURDAIN</nom>
        <naissance>2000-09-10</naissance>
        </contact>
    <contact id="6">
        <prenom>MARIE</prenom>
        <nom>DUPUY</nom>
        <naissance>1986-01-11</naissance>
    </contact>
    <contact id="7">
        <prenom>LOU</prenom>
        <nom>ANDREAS</nom>
        <naissance>1861-02-12</naissance>
    </contact>
    <contact id="9">
        <prenom>Franz</prenom>
        <nom>Kafka</nom>
        <naissance>1883-07-03</naissance>
        </contact>
    <contact id="11">
        <prenom>Joe</prenom>
        <nom>Dalton</nom>
        <naissance>2003-12-06</naissance>
        </contact>
    </mescontacts>
```

3.26.7 Traitements avec DOM :

Ecriture de fichier XML avec DOM en utilisant des données provenant d'une Base de Données. Partons de la table films suivante :

```
CREATE TABLE IF NOT EXISTS `films` (
 `code_film` int(11) NOT NULL AUTO_INCREMENT,
 `titre_original` varchar(50) DEFAULT NULL,
 `titre_francais` varchar(50) DEFAULT NULL,
 `pays` varchar(20) DEFAULT NULL,
 `date` int(11) DEFAULT NULL,
 `duree` int(11) DEFAULT NULL,
 `couleur` varchar(10) DEFAULT NULL,
 `realisateur` int(11) DEFAULT NULL,
 `image` varchar(20) DEFAULT NULL,
 PRIMARY KEY(`code_film`)
)
```

et créons un fichier XML avec les données de cette table en utilisant DOM :

```
<?php
// avec le fichier connexion.php utilisé auparavant
require("connexion.php");
$connexion=connect_bd();
$sql="SELECT * from films limit 10";
$data=$connexion->query($sql);
if ($data){
$document = new DomDocument();
$document->preserveWhiteSpace = false;
$document->formatOutput = true;
// on crée la racine <lesfilms> et on l'insère dans le document
$lesfilms = $document->createElement('lesfilms');
$document->appendChild($lesfilms);

// On boucle  pour tous les films trouvés dans la BD:
while($unfilm=$data->fetch(PDO::FETCH_OBJ))
    {
    $film=$document->createElement('film');
    $film->setAttribute('idreal', $unfilm->realisateur);
    $lesfilms->appendChild($film);
    // on crée l'élément title et on l'ajoute à $film
    $title = $document->createElement('titre');
    $film->appendChild($title);
    // on définit le texte pour $title
    $text=$document->createTextNode(utf8_encode($unfilm->titre_
↪original));
    $title->appendChild($text);

    //crée et ajoute le realisateur a $film
    $realisateur=$document->createElement('date');
    $id=$document->createTextNode($unfilm->date);
    $realisateur->appendChild($id);
    $film->appendChild($realisateur);
  }
  $document->save('myFilms.xml');
```

(suite sur la page suivante)

(suite de la page précédente)

```
  echo "Export XML fini !";
 }

 else { echo "Aucun film dans la base !" ;}

?>
```

3.26.8 Exécution :

Creation XML avec DOM

3.26.9 Resultat :

```
<?xml version="1.0"?>
<lesfilms>
  <film idreal="7">
    <titre>Pandora and the flying Dutchman                    </
↪titre>
    <date>1951</date>
  </film>
  <film idreal="8">
    <titre>Johnny Guitar                                      </
↪titre>
    <date>1954</date>
  </film>
  <film idreal="9">
    <titre>Woman under the Influence (A)                      </
↪titre>
    <date>1974</date>
  </film>
  <film idreal="10">
    <titre>Apartment (The)                                    </
↪titre>
    <date>1960</date>
  </film>
  <film idreal="11">
    <titre>Victor/Victoria                                    </
↪titre>
    <date>1982</date>
  </film>
  <film idreal="12">
    <titre>Modern Times                                       </
↪titre>
    <date>1936</date>
  </film>
  <film idreal="13">
```

(suite sur la page suivante)

(suite de la page précédente)

```
    <titre>M&#xC3;&#xA9;pris (Le)
↪    </titre>
    <date>1963</date>
  </film>
  <film idreal="14">
    <titre>Jour de f&#xC3;&#xAA;te
↪    </titre>
    <date>1948</date>
  </film>
  <film idreal="15">
    <titre>Olvidados (Los)</
↪titre>
    <date>1950</date>
  </film>
  <film idreal="16">
    <titre>West Side Story</
↪titre>
    <date>1961</date>
  </film>
</lesfilms>
```

3.26.10 Relecture avec SimpleXML :

```
<?php

$lesfilms = simplexml_load_file('myFilms.xml');
  foreach ($lesfilms->film as $film) {
    echo "Titre :". utf8_decode($film->titre)."<br/>\n";
    foreach($film->attributes() as $a => $b) {
      echo $a, '="', $b, "\"\n";
    }
    print "Annee : {$film->annee} <br/>\n";
    }

?>
```

3.26.11 Exécution :

Lecture XML avec SimpleXML

3.26.12 Resultat brut :

```
Titre :Pandora and the flying Dutchman
 ↪<br/>
idreal="7"
Annee :  <br/>
Titre :Johnny Guitar
 ↪<br/>
idreal="8"
Annee :  <br/>
Titre :Woman under the Influence (A)
 ↪<br/>
idreal="9"
Annee :  <br/>
Titre :Apartment (The)
 ↪<br/>
idreal="10"
Annee :  <br/>
Titre :Victor/Victoria
 ↪<br/>
idreal="11"
Annee :  <br/>
Titre :Modern Times
 ↪<br/>
idreal="12"
Annee :  <br/>
Titre :Mépris (Le)
 ↪<br/>
idreal="13"
Annee :  <br/>
Titre :Jour de fête
 ↪<br/>
idreal="14"
Annee :  <br/>
Titre :Olvidados (Los)
 ↪<br/>
idreal="15"
Annee :  <br/>
Titre :West Side Story
 ↪<br/>
idreal="16"
Annee :  <br/>
```

3.27 Architecture de type MVC avec PHP

3.27.1 Problème

Lorsqu'un projet augmente, le besoin de s'organiser et de permettre plus de réutilisabilité et de lisibilité demande une certaine méthode. MVC = Modèle Vue Controleur peut être une solution

intéressante.

Nous allons commencer à nous familiariser avec les composants d'un Framework MVC et à voir l'utilité de recourir à de tels outils.

Une introduction générale à ce sujet se trouve ici (http ://symfony.com/doc/current/book/from_flat_php_to_symfony2.html)

3.27.2 Du PHP pur aux templates PHP :

Considérons le code suivant en interrogeant la table CARNET vue précemment depuis PHP avec PDO :

```
<?php
  require("connect.php");
  $dsn="mysql:dbname=".BASE.";host=".SERVER;
  try{
  $connexion=new PDO($dsn,USER,PASSWD);
  }
  catch(PDOException $e){
  printf("Echec connexion : %s\n", $e->getMessage());
  exit();
  }
  $sql="SELECT * from CARNET";
  if(!$connexion->query($sql))
  echo "Pb pour acceder au CARNET";
  else
  {
      foreach ($connexion->query($sql) as $row){
          echo $row['NOM']<br/>\n";
      }
  }
?>
```

On peut observer quelques défauts dans le code ci-dessus :

- Réutilisabilté du code très réduite
- Si on fabrique un formulaire avec les entrées du carnet, où doit-on mettre le code correspondant ?

3.27.3 Un template PHP :

On peut améliorer un peu les choses :

```
<?php
   require("connect.php");
   $dsn="mysql:dbname=".BASE.";host=".SERVER;
   try
   {
```

(suite sur la page suivante)

(suite de la page précédente)

```
        $connexion=new PDO($dsn,USER,PASSWD);
    }
    catch(PDOException $e)
    {
        printf("Echec connexion : %s\n", $e->getMessage());
        exit();
    }
    $sql="SELECT * from CARNET";
    if(!$connexion->query($sql))
        echo "Pb pour acceder au CARNET";
    else
    {
        $amis=Array();
        foreach ($connexion->query($sql) as $row){
            $amis[]=$row;
        }
        require "templates/listeamis.php";
    }
 ?>
```

Avec un template listeamis.php à placer dans templates/listeamis.php :

```
<!DOCTYPE html>
<html>
<head>
    <title>Liste de mes Amis</title>
</head>
<body>
    <h1>List of friends</h1>
    <ul>
        <?php foreach ($amis as $ami): ?>
        <li>
        <a href="/recherche?nom=<?php echo $ami['ID'] ?>">
        </a>
        </li>
        <?php endforeach; ?>
    </ul>
</body>
</html>
```

On commence ainsi à séparer la présentation du codage « métier ».

3.27.4 Isolons la logique applicative :

```
<?php
  //modele.php
  require("connect.php");
```

(suite sur la page suivante)

(suite de la page précédente)

```
  function connect_db()
  {
      $dsn="mysql:dbname=".BASE.";host=".SERVER;
  try
  {
      $connexion=new PDO($dsn,USER,PASSWD);
  }
  catch(PDOException $e)
  {
  printf("Echec connexion : %s\n",
  $e->getMessage());
  exit();
  }
  return $connexion;
  }
  // Puis
  function get_all_friends()
  {
      $connexion=connect_db();
      $amis=Array();
      $sql="SELECT * from CARNET";
      foreach ($connexion->query($sql) as $row)
      {
          $amis[]=$row;
      }

      return $amis;
  }
  ?>
```

On peut maintenant avoir un **controleur** très simple qui interroge le modèle puis passe les données au template pour affichage.

```
<?php
  //c-list.php
  require_once 'modele.php';

  $amis = get_all_friends();

  require 'templates/listamis.php';
?>
```

3.27.5 Layout :

Il reste une partie non réutilisable dans le code à savoir le layout. Essayons de remédier à ça :

```
<!-- templates/baseLayout.php -->
<!DOCTYPE html>
```

(suite sur la page suivante)

(suite de la page précédente)

```
<html>
<head>
    <title><?php echo $title ?></title>
</head>
<body>
    <?php echo $content ?>
</body>
</html>
```

3.27.6 Héritage de templates :

```
<?php
// templates/t-list.php
  $title = 'Liste des amis';
  ob_start();
?>
  <h1>List de mes amis</h1>
  <ul>
  <?php foreach ($amis as $ami): ?>
  <li>
      <a href="/recherche?nom=<?php echo $ami['nom'] ?>">
          <?php echo $ami['VILLE'] ?>
      </a>
  </li>
  <?php endforeach; ?>
  </ul>
<?php
  $content = ob_get_clean();
  include 'baseLayout.php'
?>
```

Observez l'utilisation de la bufferisation avec *ob_start()* et *ob_get_clean()*. Cette dernière fonction récupère le contenu bufferisé et nettoie ensuite le buffer.

Affichage des détails d'une personne

On va ajouter à notre modèle une fonction pour afficher les détails d'une personne :

```
<?php
  function get_friend_by_id($id)
  {
      $connexion=connect_bd();
      $sql="SELECT * from CARNET where ID=:id";
      $stmt=$connexion->prepare($sql);
      $stmt->bindParam(':id', $id, PDO::PARAM_INT);
      $stmt->execute();
```

(suite sur la page suivante)

(suite de la page précédente)

```
        return $stmt->fetch();
    }
```

On peut maintenant créer un nouveau controleur c-details.php :

```
<?php
  //c-details.php
  require_once 'modele.php';
  $pers = get_friend_by_id($_GET['id']);
  require 'templates/t-details.php';
?>
```

Qui utilise le template :

```
<?php
  //templates/t-details.php
  $title = $pers['NOM'];
  ob_start();
?>
<h1>details sur
<?php echo $pers['PRENOM'].' '.$pers['NOM'] ?>
</h1>
<p>
<?php
  echo ' Ne le '.$pers['NAISSANCE'];
  echo '<br/>Ville:'.$pers['VILLE'];
  $content = ob_get_clean();
  include 'baseLayout.php'
?>
```

Vous pouvez tester en entrant l'URL de c-details.php avec un paramètre id. Le code est similaire à celui du premier template et nous pouvons réutiliser le template de base, mais il subsiste plusieurs problèmes :

— Si le paramètre id n'est pas fourni, notre application va provoquer une erreur.
— Nous n'avons pas de controleur principal.

Regroupons d'abord le code des 2 contrôleurs (c-list.php et c-details.php) dans un fichier unique controllers.php

```
<?php
// controllers.php
  function list_action()
  {
    $amis = get_all_friends();
    require 'templates/t-list.php';
  }

  function detail_action($id)
  {
   $pers = get_friend_by_id($id);
```

(suite sur la page suivante)

(suite de la page précédente)

```
    require 'templates/t-detail.php';
  }
?>
```

Nous pouvons enfin proposer un controleur principal (Front Controller) `index.php` :

```
<?php
  // index.php
  // On charge les modeles et les controleurs
  require_once 'modele.php';
  require_once 'controllers.php';
  // gestion des routes
  $uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
  if ('/index.php' == $uri)
    {
        list_action();
    }
  elseif ('/index.php/detail' == $uri && isset($_GET['id']))
    {
        detail_action($_GET['id']);
    }
    else
    {
        header('Status: 404 Not Found');
        echo '<html><body><h1>Page Not Found</h1></body></html>';
    }
  ?>
```

Nous avons maintenant une structure de ce type :

```
├── connect.php
├── connexion.php
├── controlleur.php
├── modele.php
├── recherche.php
└── templates
    ├── layout.php
    └── listeamis.php
```

On peut améliorer tout cela en intégrant dans un même Objet tout le modèle. Voir sur github/roza/php-basic-mvc (https ://github.com/roza/php-basic-mvc/) pour le code complet. Mais ce système de routage est encore très incomplet et nous allons plutôt utliser pour continuer les outils d'un framework Web de référence : Symfony.

3.28 Templates Twig en PHP

L'installation de Twig se fait grâce à l'outil *composer*

3.28.1 composer

Installons d'abord l'outil composer qui servira à beaucoup de tâches courantes dans un projet PHP. On va d'abord créer un répertoire `bin` à la racine de notre `HOME`, placez-y l'exécutable `composer.phar` et renommez le `composer`.

```
cd
mkdir bin
cd bin
curl -s https://getcomposer.org/installer | php
mv composer.phar composer
```

Vérifiez la définition de vos variables d'environnement `http_proxy` et `https_proxy` dans votre `.bashrc` Ajoutez également la ligne suivante à votre `.bashrc` :

```
export PATH=$PATH:~/bin
```

de manière à ce que tous les programmes installés dans le répertoire *bin* de votre HOME soient accessibles de n'importe où.

3.28.2 Installation

Installons Twig : .. code-block : : none

composer require twig/twig :~1.0

Ceci créera dans le répertoire courant un dossier **vendor** contenant les librairies demandées.

On définit d'abord un template de base, BaseTemplate.html :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    {% block head %}
    <meta charset="utf-8">
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %}</title>
    {% endblock %}
</head>
<body>
<section id="content">
{% block content %}{% endblock %}
</section>
<footer id="footer">
    {% block footer %}
    &copy; Copyright 2018 <a href="http://monsite.com">
    Mon super Site</a>.
    {% endblock %}
</footer>
</body>
</html>
```

Puis un template plus spécialisé qui en hérite, menu.html :

```
{% extends "BaseTemplate.html" %}
{% block title %}Menu de la semaine{% endblock %}
{% block head %}
{{ parent() }}
<style type="text/css">
    .important { color: #336699; }
</style>
{% endblock %}

{% block content %}
<h1>Menu</h1>
<p class="important">
    Voici votre menu de la semaine:
    <dl>
        <dt>Lundi</dt>
        <dd>{{Lundi}}</dd>
        <dt>Mardi</dt>
        <dd>{{Mardi}}</dd>
        <dt>Mercredi</dt>
        <dd>{{Mercredi}}</dd>
        <dt>Jeudi</dt>
        <dd>{{Jeudi}}</dd>
    </dl>
</p>
{% endblock %}
```

Enfin, on utilise ce template dans un fichier menu.php en chargeant d'abord l'"**autoloader** :

```
<?php
// inclure  l'autoloader
include 'vendor/autoload.php';

try {
    // Définir le dossier ou on trouve les templates
    $loader = new Twig_Loader_Filesystem('templates');
    // initialiser l'environement Twig
    $twig = new Twig_Environment($loader);
    // charger le template
    $template = $twig->loadTemplate('menu.html');
    // Affecter les variables du template et appeller le rendu
    echo $template->render(array(
        'Lundi' => 'Steak Frites',
        'Mardi' => 'Raviolis',
        'Mercredi' => 'Pot au Feu',
        'Jeudi' => 'Couscous',
        'Vendredi' => 'Poisson'
    ));
} catch (Exception $e) {
    die ('ERROR: ' . $e->getMessage());
```

(suite sur la page suivante)

(suite de la page précédente)

```
}
```

3.28.3 Affichage des personnes du Carnet

Affichons à présent les personnes du Carnet à l'aide d'un template Twig. On réutilise le fichier modele.php vu précédemment :

```
<?php
// modele.php
class Carnet {
    private static $connexion;
    function __construct(){
        $dsn="mysql:dbname=".BASE.";host=".SERVER;
        try{
                self::$connexion=new PDO($dsn,USER,PASSWD);
            }
        catch(PDOException $e){
            printf("Échec de la connexion : %s\n", $e->
↪getMessage());
            $this->connexion = NULL;
        }
    }
    /** Récupére la liste des contacts sous forme d'un tableau */
    function get_all_friends(){
        $sql="SELECT * from CARNET";
        $data=self::$connexion->query($sql);
        return $data;
    }
    /** Ajoute un contact à la table CARNET */
    function add_friend($data){
        $sql = "INSERT INTO CARNET(NOM,PRENOM,NAISSANCE,VILLE)␣
↪values (?,?,?,?)";
        $stmt = self::$connexion->prepare($sql);
        return $stmt->execute(array($data['nom'],
            $data['prenom'], $data['naissance'],$data['ville']));
    }
    /** Récupére un contact à partir de son ID */
    function get_friend_by_id($id)
    {
        $sql="SELECT * from CARNET where ID=:id";
        $stmt=self::$connexion->prepare($sql);
        $stmt->bindParam(':id', $id, PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetch(PDO::FETCH_OBJ);
    }
```

```
<?php
```

(suite sur la page suivante)

(suite de la page précédente)

```
// fichier carnet.php
include 'vendor/autoload.php';

// on inclus le modele
include 'modele.php';
// On instancie un Carnet
$car = new Carnet();
try {
    // Indiquer le dossier ou on trouve les templates
    $loader = new Twig_Loader_Filesystem('templates');
    // initialiser l'environement Twig
    $twig = new Twig_Environment($loader);

    // Récupérer les données depuis la base
    $amis = $car->get_all_friends();

    // Charger le template
    $template = $twig->loadTemplate('carnet.html');
    $titre="Liste des membres du carnet trié par ville";
    echo $template->render(array(
        'titre' => $titre,
        'amis' => $amis,
        ));

} catch (Exception $e) {
    die ('ERROR: ' . $e->getMessage());
}
?>
```

et un template carnet.html :

```
{% extends "BaseTemplate.html" %}
{% block title %}Personnes du Carnet {% endblock %}
{% block head %}
{{ parent() }}
<style type="text/css">
    .important { color: #336699; }
</style>
{% endblock %}

{% block content %}

<p align="center" class="Style1">{{titre}}</p>
<table border="2" align="center" cellspacing="0" cellpadding="2" >
<tr bgcolor="#CA9999">
    <td width="50"><strong>numero</strong></td>
    <td width="50"><strong>Nom</strong></td>
    <td width="50"><strong>Prenom</strong></td>
    <td width="30"><strong>Age</strong></td>
```

(suite sur la page suivante)

(suite de la page précédente)

```
    <td width="50"><strong>Ville</strong></td>
</tr>
{% set i=0 %}
{% for ami in amis %}
{% set i=i+1 %}
{% if i is odd %}
<tr bgcolor="#F0F0F0">
{% else %}
    <tr bgcolor="#A6A6A6">
{% endif %}
    <td>{{ami.ID}}</td>
    <td>{{ami.NOM}}</td>
    <td>{{ami.PRENOM}}</td>
    <td>{{ami.NAISSANCE}}</td>
    <td>{{ami.VILLE}}</td>
</tr>
{% endfor %}
</table>

{% endblock %}
```

Ce template est un peu maladroit mais il montre l'expressivité du langage de template Twig avec des boucles, des conditionnelles, calculs avec des variables, etc.

Nous pouvons bien sûr en proposer un plus simple avec le CSS adéquat.

3.28.4 Affichage des Personnes avec un template plus simple :

(carnet2.html)

```
{% extends "BaseTemplate.html" %}
{% block title %}Personnes du Carnet{% endblock %}
{% block head %}
{{ parent() }}
<link rel="stylesheet" href="static/css/tabstyle.css" />
{% endblock %}

{% block content %}

<h2>{{titre}}</h2>
<table id="jolie" class="centre" >
<tr>
    <td>numero</td>
    <td>Nom</td>
    <td>Prenom</td>
    <td>Age</td>
    <td>Ville</td>
</tr>
```

(suite sur la page suivante)

(suite de la page précédente)

```
{% for ami in amis %}
    <tr>
    <td>{{ami.ID}}</td>
    <td>{{ami.NOM}}</td>
    <td>{{ami.PRENOM}}</td>
    <td>{{ami.NAISSANCE}}</td>
    <td>{{ami.VILLE}}</td>
</tr>
{% endfor %}
</table>

{% endblock %}
```

avec le style qui va bien sur les tableaux ...

Nous pouvons ainsi compléter le développement MVC effectué précédemment en utilisant des templates Twig. Voir sur github/roza/php-basic-mvc (https ://github.com/roza/php-basic-mvc/) pour un code plus complet. Mais le système de routage employé est encore très rudimentaire et nous allons plutôt utliser les outils d'un framework Web de référence : Symfony 4 pour aller plus loin.

3.29 Composer, Symfony 4 et Flex

Nous allons à présent nous familiariser avec les outils et composants d'un Framework de référence : Symfony 4 qui est très modulaire et permet d'installer des composants très riches comme SwiftMailer pour envoyer des mails, FOSUserBundle pour gérer des utilisateurs, FOSRESTBundle ou APIPlatform pour réaliser rapidement une API complète. Le Framework Symfony 4 est basé sur un Micro-noyau (Micro-Kernel) 70% plus léger que le noyau de Symfony 3.

Une introduction générale à ce framework se trouve ici (https ://symfony.com/4)

La gestion des dépendances se fait à présent grâce à l'outil Symfony *Flex* qui permet d'établir des recettes ou *recipes* décrivant les dépendances et la configuration d'un projet. L'outil de base est *composer*

Indication : Composer permet d'installer des centaines de packages librement disponibles. On les trouve sur Packagist (https ://packagist.org/) . Il permet de gérer les dépendances d'un projet et également de créer le squellette d'une application Symfony 4.

3.29.1 composer

Installons d'abord l'outil composer qui servira à beaucoup de tâches courantes dans un projet PHP. On va d'abord créer un répertoire `bin` à la racine de notre `HOME`, placez-y l'exécutable `composer.phar` et renommez le `composer`.

```
cd
mkdir bin
cd bin
curl -s https://getcomposer.org/installer | php
mv composer.phar composer
```

Vérifiez la définition de vos variables d'environnement `http_proxy` et `https_proxy` dans votre `.bashrc` Ajoutez également la ligne suivante à votre `.bashrc` :

```
export PATH=$PATH:~/bin
```

de manière à ce que tous les programmes installés dans le répertoire *bin* de votre HOME soient accessibles de n'importe où.

Si on veux juste installer un composant simple comme *HTTP Foundation », on place à la racine du dossier de travail le fichier *composer.json* suivant :

```
{
"require": {
    "symfony/http-foundation": "~4.0"
    }
}
```

Ceci indique que nous n'installons pour l'instant que ce seul composant et que nous demandons la dernière version stable de la branche **4** pour http-foundation. Puis utilisons `composer` pour installer les composants demandés :

```
composer update -o
```

Notez l'utilisation de l'option -o pour *optimize-autoloader* qui optimise « au mieux » le chargement automatique des classes.

3.29.2 HttpFoundation :

Les 2 principaux composants de HttpFoundation à savoir `Request` et `Response` sont alors prêts à l'emploi.

Remarquez l'usage des espaces de nommages en PHP semblables à ceux du C++ ou aux import de packages en java pour éviter des conflits de nommages entre différents *vendor* c'est à dire différentes entités fournissant du code à votre projet :

```
<?php
  // chargement autoloader
  require_once __DIR__.'vendor/autoload.php';

  use Symfony\Component\HttpFoundation\Request;
  use Symfony\Component\HttpFoundation\Response;
  // Actual request :
  //$request = Request::createFromGlobals();
```

(suite sur la page suivante)

(suite de la page précédente)

```
  // fake request
  $request = Request::create('/essai.php?name=Zozo');

  // URI demandee (sans les parametres)
  $path=$request->getPathInfo();

 // recup GET de variables
  $nom=$request->query->get('name','World');
  $prenom=$request->query->get('surname','Joe');
  echo "Bonjour $surname $name<br/>";
```

On peut aussi récupérer d'autres informations sur le Client et fabriquer une réponse :

```
<?php

// recup  variables SERVER
$host=$request->server->get('HTTP_HOST');

// get  COOKIES
$request->cookies->get('PHPSESSID');

// HTTP headers
$headers=$request->headers->get('host');
$content_type=$request->headers->get('content_type');

$method=$request->getMethod();  //GET, POST, PUT, DELETE ou HEAD
$langs=$request->getLanguages();
$IP == $request->getClientIp();
$response =
new Response($IP." ".$host." ".$path." ".$headers."
".$content_type." ".$method." ".$nom." ".$langs[0]);
$response->send();
?>
```

3.29.3 Squellette d'application Symfony 4 :

Créons un répertoire de travail mvc-sf4 dans votre dossier Web, par exemple ~/www (ou ~/public_html) si on utilise user_dir d'Apache ou n'importe où si on utilise le serveur Web embarqué de PHP.

et créons la trame d'une application symfony4 à l'aide de composer

```
composer create-project symfony/skeleton hello-sf4
```

Veillez à bien avoir une version à jour de composer et si besoin :

```
composer self-update
```

La structure du projet ainsi créé est la suivante :

```
hello-sf4/
├── bin
│   └── console
├── composer.json
├── composer.lock
├── config
│   ├── bundles.php
│   ├── packages
│   ├── routes.yaml
│   └── services.yaml
├── public
│   └── index.php
├── src
│   ├── Controller
│   └── Kernel.php
├── symfony.lock
├── var
│   ├── cache
│   └── log
└── vendor
    ├── autoload.php
    ├── composer
    ├── psr
    └── symfony
```

Le répertoire bin contient l'outil console qui permet d'effectuer les tâches de routine pour créer ou gérer un projet. Le répertoire config contient les fichiers de configuration. Le répertoire public contient le fichier index de l'application Le dossie src les controleurs, le Kernel mais aussi les entités etc. Le dossier var contient les cache et les logs et le dossier vendor les classes des Bundles installés comme http-foundation.

Vous pouvez consulter le fichier *symfony.lock* qui se trouve à la racine du dossier hello-sf4 pour voir la liste des dépendances installées :

```
{
    "psr/cache": {
        "version": "1.0.1"
    },
    "psr/container": {
        "version": "1.0.0"
    },
    "psr/log": {
        "version": "1.0.2"
    },
    "psr/simple-cache": {
        "version": "1.0.1"
    },
    "symfony/cache": {
        "version": "v4.1.0"
    },
```

(suite sur la page suivante)

(suite de la page précédente)

```
    "symfony/config": {
        "version": "v4.1.0"
    },
    "symfony/console": {
        "version": "3.3",
        "recipe": {
            "repo": "github.com/symfony/recipes",
            "branch": "master",
            "version": "3.3",
            "ref": "e3868d2f4a5104f19f844fe551099a00c6562527"
        }
    },
    "symfony/debug": {
        "version": "v4.1.0"
    },
    "symfony/dependency-injection": {
        "version": "v4.1.0"
    },
    "symfony/dotenv": {
        "version": "v4.1.0"
    },
    "symfony/event-dispatcher": {
        "version": "v4.1.0"
    },
    "symfony/filesystem": {
        "version": "v4.1.0"
    },
    "symfony/finder": {
        "version": "v4.1.0"
    },
    "symfony/flex": {
        "version": "1.0",
        "recipe": {
            "repo": "github.com/symfony/recipes",
            "branch": "master",
            "version": "1.0",
            "ref": "cc1afd81841db36fbef982fe56b48ade6716fac4"
        }
    },
    "symfony/framework-bundle": {
        "version": "3.3",
        "recipe": {
            "repo": "github.com/symfony/recipes",
            "branch": "master",
            "version": "3.3",
            "ref": "c0c0bf94174609d740ca2a61e0201949c5683d50"
        }
    },
    "symfony/http-foundation": {
```

(suite sur la page suivante)

(suite de la page précédente)

```
        "version": "v4.1.0"
    },
    "symfony/http-kernel": {
        "version": "v4.1.0"
    },
    "symfony/lts": {
        "version": "4-dev"
    },
    "symfony/polyfill-mbstring": {
        "version": "v1.8.0"
    },
    "symfony/routing": {
        "version": "4.0",
        "recipe": {
            "repo": "github.com/symfony/recipes",
            "branch": "master",
            "version": "4.0",
            "ref": "cda8b550123383d25827705d05a42acf6819fe4e"
        }
    },
    "symfony/yaml": {
        "version": "v4.1.0"
    }
}
```

3.29.4 Application simple Symfony 4

Complétons à présent l'application Symfony 4.

3.30 Tester une application PHP - TDD

Nous allons à présent nous attaquer à une problématique fondamentale dans toute application qu'elle soit Web, mobile ou autres : Les tests.

3.30.1 TDD

TDD veut dire *Test Driven Development* c'est à dire *Développement dirigé par les tests* C'est une démarche mise en avant en *Méthodologie Agile* Elle consiste en général en l'application des points suivants :

— écrire un test
— vérifier qu'il échoue (car le code qu'il teste n'existe pas)
— écrire juste le code suffisant pour passer le test
— vérifier que le test passe

— procéder à un refactoring du code, c'est-à-dire l'améliorer en gardant les mêmes fonctionnalités.

3.30.2 Intérêt de la démarche :

Les avantages principaux de cette démarche sont :
— Préciser au mieux les spécifications du code et l'API envisagée
— Ceci oblige à faire des choix de conception qui restent parfois trop dans le flou au début du développement
— Plus tard, disposer d'une large base de tests est une riche pour une application car elle permet de vérifier à tout moment que les tests installés ne sont pas mis en défaut par de nouveaux développements ou des refactoring de code

Tous les langages de programmation disposent de Frameworks de tests. Par exemple Java offre JUnit.

PHP quand a lui propose PHPUnit. On peut l'installer via composer :

```
{
"require": {
    "phpunit/phpunit": "6.3.*",
},
"autoload": {
    "psr-0": {
        "Exemple": "src"
        }
    }
}
```

```
composer.phar install
```

Ecrivons à présent notre premier test dans le dossier Tests :

```
<?php
use Exemple\FileLoader;

class FileLoaderTest extends PHPUnit_Framework_TestCase
{
    public function testFileLoaderClassCanBeCreated()
    {
        $f = new FileLoader;
    }
}
```

Pour tester : Placer un fichier *phpunit.xml* à la racine de votre projet contenant :

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
```

(suite sur la page suivante)

(suite de la page précédente)

```
PHPUnit Configuration
=====================
Fichier de configuration standard de phpunit
-->
<phpunit backupGlobals="false"
         backupStaticAttributes="false"
         colors="true"
         bootstrap="vendor/autoload.php"
         convertErrorsToExceptions="true"
         convertNoticesToExceptions="true"
         convertWarningsToExceptions="true"
         processIsolation="false"
         stopOnFailure="false"
         syntaxCheck="false"
>
    <testsuites>
        <testsuite>
            <directory>tests</directory>
        </testsuite>
    </testsuites>
</phpunit>
```

Nous sommes prêts à lancer notre premier test :

```
phpunit
```

Ecrivons maintenant un peu de code pour nous permettre de passer notre premier test :

Nous allons compléter notre test par une vérification qu'un fichier situé dans les textit{fixtures} de test a bien été chargé :

```
<?php
class FileLoaderTest extends PHPUnit_Framework_TestCase
{
    public function testFileLoaderClassCanBeCreated()
    {
        $f = new FileLoader;
    }

    /**
    * Nous voulons récupérer le contenu d'un fichier via
    * une méthode get()
    */
    public function testFileLoaderCanLoadFileContent()
    {
        $f = new FileLoader;
        $r = $f->get(__DIR__.'/fixtures/simple.md');
        $this->assertEquals("Foo\n", $r);
    }
}
```

Puis si nous avons besoin de *Mock Objects*, nous installerons la librairie *mockery* :

```
composer require --dev mockery/mockery
```

3.31 Mettre en place un Web Service REST

3.31.1 Problème

Dans une architecture REST classique, un serveur présente les données d'une table et un Client riche (ou RIA) en JavaScript ou un client Mobile permet de les récupérer et des les afficher. REST signifie *Representational State Transfer*.

Cette architecture permet de réaliser des applications de type *onepage* en reportant sur le client une bonne partie de la logique métier et en offrant des point d'entrée aux clients pour lire des données sur le serveur ou lui en envoyer.

Ces données pourront être envoyées en XML ou de plus en plus aujourd'hui en JSON : JavaScript Object Notation, c'est à dire des objets directement utilisables en JS.

On pose les définitions suivantes :

- RIA = Rich Internet Application
- REST = Representational State Transform
- Logique métier déportée vers le client
- Tâche principale du serveur : Offrir des services de récupération et de stockage de données

Un flux de news pourra ainsi offrir par exemple une ressource du type : /api/v1/news/314159 qui permettra aux clients de récupérer la news numéro 314159 en JSON ou en XML en employant la méthode HTTP GET dans la version 1 de notre API. Dans cet exemple, la news est ici la ressource ou *élément* manipulée dans l'API version 1. La méthode GET sera employée pour récupérer des éléments *individuellement* ou par *Collections*.

La méthode POST sera quand à elle employée pour envoyer vers le serveur un ou plusieurs éléments. D'autres méthodes HTTP pour créer ou modifier complètement (PUT) ou partiellement (PATCH) des éléments ou les effacer (DELETE) seront souvent également disponibles dans l'API.

Les technologies concurrentes à REST sont XML-RPC et SOAP (Microsoft) REST est une façon moderne de concevoir ce genre de service et possède les avantages suivants :

- Bonne montée en charge du serveur
- Simplicité des serveurs (retour aux sources du protocole HTTP)
- Equilibrage de charge
- le serveur offre une API
- les services sont représentés par des URL's donc simplicité et bonne gestion du cache
- Possibilité de décomposer des services complexes en de multiples services plus simples qui communiquent entre eux

Les principes de REST ont été théorisés par Roy Fielding dans sa thèse (http ://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm) :

1. Séparation claire entre Client et Serveur

2. Le client contient la logique métier, le serveur est sans Etat
3. Les réponses du serveur peuvent ou non être mises en cache
4. L'interface doit être simple, bien définie, standardisée
5. Le système peut avoir plusieurs couches comme des proxys, systèmes de cache, etc
6. Eventuellement, les clients peuvent télecharger du code du serveur qui s'exécutera dans le contexte du client

Pour mémoire, une API REST peut offrir les méthodes suivantes :

Méthodes HTTP et REST :

Méthode	Rôle	Code retour HTTP
GET URL	Récupération Element	200
GET URL	Récupération Collection	201
POST URL	Envoi d'Elements	201
DELETE URL	Effacer Element(s)	200
PUT URL	Modifier un Element	200
PATCH URL	Modif. partielle d'Elt.	200

Mais on peut aussi avoir des erreurs comme :

Code Erreur	Description	Signification
400	Bad Request	requête mal formée
404	Not Found	Resource demandée inexistante
401	Unauthorized	Authentification necessaire pour accéder à la resource.
405	Method Not Allowed	Méthode interdite pour cette resource.
409	Conflict	Par exemple, un PUT qui crée une ressource 2 fois
500	Internal Server Error	Autres erreurs du serveur.

Par ailleurs, le serveur REST ne maintient pas d'état, les requêtes sont indépendantes les unes des autres. C'est un retour aux fondamentaux du protocole HTTP qui n'est pas doté de beaucoup de capacités de mémorisation ...

La logique et l'ergonomie de l'application sont gérées côté client. C'est une méthode aujourd'hui plebiscitée pour faire dialoguer des clients (mobiles ou Web) avec des serveurs.

3.32 Exemple de service REST avec PHP

3.32.1 Problème

Nous allons réaliser en PHP l'implémentation d'un service REST qui exposera les données de la table de contact appellée CARNET utilisée dans les autres exemples.

Un contact sera ainsi ainsi accessible à une route du type : /api/v1/contact/12 qui permettra aux clients de récupérer le contact en JSON employant la méthode HTTP GET dans la version 1 de notre API. Dans cet exemple, le contact constitue la *ressource* manipulée dans notre API. La méthode GET sera employée pour récupérer des éléments *individuellement* ou par *Collections*.

Méthode	Action réalisée	URI
GET	Récup. tous les liens	/api/v1/
GET	Récuperation un Element	/api/v1/contact/{id}
GET	Récupération Collection	/api/v1/contact
POST	Creation d'Elements	/api/v1/contact
DELETE	Effacer Element	/api/v1/contact/{id}
PUT	Modifier un Element	/api/v1/contact/{id}
PATCH	Modif. partielle d'Elt.	/api/v1/contact/{id}

La route /api/v1/ en GET renverra la liste des URLs des contacts plutôt que la liste de tous les contacts avec tous leurs détails. Ceci permet d'avoir un serveur REST auto-documenté où la récupération d'une première URL permet en suivant d'obtenir la liste des ressources présentes sur le service avec leurs URLs respectives.

On pourra également paginer les réponses pour ne pas manipuler trop de données simultanément.

Pour assurer le routage simplement nous allons continuer avec Silex (http ://silex.sensiolabs.org/)

Nous pouvons donc modifier le fichier index.php déjà mis en place comme suit :

```
<?php
require_once __DIR__.'/vendor/autoload.php';
require_once 'modele.php';

$app = new Silex\Application();
$app['debug']=true;

$app->get('/contact', function () {
    $content ='<ul>';
    $amis=get_all_friends();
    foreach ($amis as $ami){
        $content.='<li>'.$ami['NOM'].'</li>';
    }
    $content.='</ul>';
    return $content;
});

$app->get('/api/', function () {
    $amis=get_all_friends_links();
return json_encode($amis);
});

$app->get('/api/contact', function () {
```

(suite sur la page suivante)

(suite de la page précédente)

```
    $amis=get_all_friends();
    return json_encode($amis);
});
?>
```

avec une nouvelle méthode dans modele.php :

```
<?php
  function get_all_friends_links()
  {
       $connexion=connect_db();
       $amis=Array();
       $sql="SELECT * from CARNET";
       $data=$connexion->query($sql);
       while($pers=$data->fetch(PDO::FETCH_ASSOC))
         {
              $res=Array();
              $res['NOM'] = $pers['NOM'];
              $res['URL']=$_SERVER["REQUEST_SCHEME"].'://'.
                          $_SERVER['HTTP_HOST'].
                          $_SERVER['CONTEXT_PREFIX'].
                          '/silex/api/contact/'.$pers['ID'];
              $amis[] = $res;
         }
       return $amis;
  }
  ?>
```

Indication : La vue de base de notre API renvoie maintenant la liste des liens de nos contacts et quelqu'un qui s'y connecte pourra décrouvrir par la d'autres URLs gérées par notre API. Une bonne API REST se doit d'être **autodocumentée** dans la mesure du possible !

Puis assurons le GET sur l'URI /api/contact/id en ajoutant à index.php :

```
<?php
    $app->get('/api/contact/{id}', function($id) use ($app) {
    $ami = get_friend_by_id($id);
    if (!$ami) $app->abort(404, "Contact inexistant");
    else return json_encode($ami,JSON_PRETTY_PRINT);
    });
?>
```

qui marchera si on ajoute la nouvelle méthode get_friend_by_id() au modèle :

```
<?php
    function get_friend_by_id($id)
    {
```

(suite sur la page suivante)

(suite de la page précédente)

```
      $connexion=connect_db();
      $sql="SELECT * from CARNET where ID=:id";
      $stmt=$connexion->prepare($sql);
      $stmt->bindParam(':id', $id, PDO::PARAM_INT);
      $stmt->execute();
      return $stmt->fetch(PDO::FETCH_OBJ);
    }
?>
```

Continuons avec la méthode http DELETE sur la même route en ajoutant à index.php :

```
<?php
    $app->delete('/api/contact/{id}', function($id) use ($app) {
    $ami = get_friend_by_id($id);
    if (!$ami)
        $app->abort(404, "Contact inexistant");
    else {
        delete_friend_by_id($id);
        return json_encode($ami,JSON_PRETTY_PRINT);
    }
    });
?>
```

en ajoutant au modèle :

```
<?php
    function delete_friend_by_id($id)
    {
      $connexion=connect_db();
      $sql="Delete from CARNET where ID=:id";
      $stmt=$connexion->prepare($sql);
      $stmt->bindParam(':id', $id, PDO::PARAM_INT);
      $stmt->execute();
      return $stmt->fetch(PDO::FETCH_OBJ);
    }
?>
```

Enfin le POST doit nous permettre d'envoyer un nouveau contact pour peupler la table CARNET sur la route /api/contact. Nous assurons d'abord la récupération du contenu json sous la forme d'un tableau PHP avec la méthode before de Silex :

```
<?php
        $app->before(function (Request $request) {
            if (0 === strpos($request->headers->get('Content-Type'),
↪ 'application/json'))
            {
                $data = json_decode($request->getContent(), true);
                $request->request->replace(is_array($data) ? $data␣
↪: array());
```

(suite sur la page suivante)

(suite de la page précédente)

```
            }
        });
?>
```

Puis la méthode post proprement dite :

```
<?php
    $app->post('/api/contact', function (Request $request) use (
↪$app) {
      $data = $request->request->all();
      add_friends($data);
      return new Response(json_encode($data), 200, array('Content-
↪Type' => 'application/json'));
    });
?>
```

N'oubliez pas de faire appel aux objets Request et Response au début du fichier index.php :

```
<?php
  use Symfony\Component\HttpFoundation\Request;
  use Symfony\Component\HttpFoundation\Response;
?>
```

Il ne reste plus qu'à ajouter au modèle :

```
<?php
    function add_friends($data)
    {
      $connexion=connect_db();
      $sql="INSERT INTO CARNET(NOM,PRENOM,NAISSANCE,VILLE) values (?
↪,?,?,?)";
      $stmt=$connexion->prepare($sql);
      return $stmt->execute(array($data['NOM'], $data['PRENOM'],
↪$data['NAISSANCE'],$data['VILLE']));
    }
?>
```

Il n'y a plus qu'à implémenter un PUT et surtout à Tester ! !

3.33 Tester une API REST avec votre navigateur ou avec curl

Pour tester notre API nous pouvons dans un premier temps utiliser l'extension **Postman** de Chrome ou l'extension **RESTClient** pour Firefox.

Avertissement : Attention à bien désactiver les proxys dans vos navigateurs si vous voulez utiliser ces extensions. Pour chrome on peut le lancer en ligne de commande avec l'option –no-proxy-server

Si on veut une solution en ligne de commande, *curl* permet de manipuler les différentes méthodes HTTP. La syntaxe n'est pas idéale mais on peut tester toutes les méthodes HTTP.

Avertissement : Si vous avez déclaré des variables d'environnement http_proxy ou https_proxy, il vaut mieux les désactiver pour que curl n'essaie pas de passer par un proxy ce qui serait problématique pour certaines de ces requêtes curl :

```
unset http_proxy
unset https_proxy
```

3.33.1 Pour tester un GET :

```
curl -i http://localhost/silex/api/v1/contact/2
```

ou si on utilise le module user_dir d'Apache.

```
curl -i http://localhost/~login/silex/api/v1/contact/2
```

Avertissement : Attention dans le cas ou on utilise *user_dir*, les réglages pour utiliser toutes les methodes du protocole HTTP/1.1 peuvent s'avérer délicats.

Notamment le fichier /etc/apache2/mods-available/userdir.conf doit être modifié comme suit (les user_dir sont dans www, syntaxe pour Apache > 2.2) :

```
<IfModule mod_userdir.c>
    UserDir www
    UserDir disabled root

    <Directory /home/*/www>
            AllowOverride All
            Options MultiViews Indexes SymLinksIfOwnerMatch
            <Limit GET POST PUT DELETE OPTIONS>
                    Require all granted
            </Limit>
            <LimitExcept GET POST PUT DELETE OPTIONS>
                    Require all denied
            </LimitExcept>
    </Directory>
</IfModule>
```

et il faut aussi dans votre dossier sous voter user_dir (par exemple ~/www/silex) avoir le .htaccess suivant :

```
<Limit GET POST DELETE PUT OPTIONS>
allow from all
</Limit>

FallbackResource /~roza/silex/index.php
RewriteBase /~roza/silex
```

Dans **tous les cas**, préciser si besoin dans votre php.ini ou un fichier équivalent :

```
always_populate_raw_post_data = -1
```

3.33.2 Pour tester un POST :

```
curl -i -H "Content-Type: application/json" -X POST
 -d '{"NOM":"Dalton", "PRENOM":"joe", "NAISSANCE":"2000-08-15",
↪"VILLE":"Orleans"}' http://localhost/silex/api/contact
```

3.33.3 Test un PUT :

```
curl -i -H "Content-Type: application/json" -X PUT -d '{"done":true}
↪' http://localhost/silex/api/contact/5
```

3.33.4 Test de DELETE :

```
curl -i -H "Content-Type: application/json" -X "DELETE"
http://localhost/silex/api/contact/7
```

3.34 Tester une API

3.34.1 Tester une API avec Postman

C'est le moyen le plus simple pour tester une API. On l'installe en téléchargeant l'App de Postman (https ://www.getpostman.com/apps) pour votre OS. On peut alors se constituer des collections de requêtes HTTP pour tester une API REST spécifique.

3.34.2 Tester une API REST avec Guzzle

Les tests de notre API REST avec curl sont peu lisibles. Si vous préférez disposer d'une batterie de tests en PHP, vous pouvez utiliser une librairie spécialisée comme Guzzle. On peut installer cette dernière avec composer :

```
composer require guzzlehttp/guzzle
```

3.35 Feuilles de TD Lic Pro Web et Mobile

3.35.1 Année 2018-2019

— TD1
— TD2
— TD3

3.36 Feuilles de TD 2ème Année IUT informatique

3.36.1 Année 2018-2019

— TD1
— TP1
— TD2
— TP2
— TP3
— TD3
— Git init

3.37 Feuilles de TD Lic Pro Web et Mobile

3.37.1 Année 2017-2018

— TD1
— TD2
— TD3
— TD4
— TD5
— TD6
— TD7

3.38 Feuilles de TD CVRH Tours

3.38.1 Formation de Juin 2015

— TD1
— TD2
— TD3

3.39 Alice démarre avec git :

3.39.1 Paramétrage et initialisations :

On configure d'abord ses paramètres

```
git config --global user.name "Alice Torvalds"
git config --global user.email "alice@kernel.org"
```

3.39.2 Création d'un dossier local versionné

```
mkdir monprojet
cd monprojet
git init
```

Si vous avez déja du contenu :

```
git add .
```

3.39.3 Création d'un dépot « monprojet » sur gitlab

— Privé
— Public
— ou Interne à gitlab

> Bitbucket (https ://bitbucket.org/) offre également la possibilité d'avoir des dépôts privés de taille limitée.
> Github (https ://github.com/) offre les dépôts public et fait payer les dépôts privés.

3.39.4 Connexion entre le local et le gitlab :

Eventuellement : git config push.default simple

```
git remote add origin https://gitlab.com/alice/monprojet.git
git push -u origin master
```

ou simplement :

```
git push
```

par la suite

3.39.5 Réalisation d'une fonctionnalité par Alice :

— Alice prend une chose à réaliser et implémente le code nécessaire
— Alice fait les tests et vérifie que ça marche
— git commit -am « Message de commit »

3.39.6 Alice pousse son master sur son remote :

```
git push -u origin master
```

3.40 Bob travaille avec Alice grâce à git :

Bob fait d'abord comme Alice pour paramétrer et initialiser son dépôt local.

3.40.1 Bob vérifie qu'il a bien 2 remotes :

— **Le sien, origin qu'il crée au besoin en faisant :**

```
git remote add origin https://gitlab.com/bob/monprojet.git
```

— **celui d'Alice qu'il ajoute :**

```
git remote add alice https://gitlab.com/alice/monprojet.git
```

— il tape git remote -v pour vérifier ses remotes
— **Si il se trompe :**

```
git remote remove alice
```

3.40.2 Bob récupère le master d'Alice :

```
git fetch Alice master
```

3.40.3 Bob consulte la branche locale correspondant au master d'Alice :

```
git branch -av
git checkout Alice/master
```

puis vérifie que le code d'Alice est correct

3.40.4 Bob revient dans son master :

```
git checkout master
```

3.40.5 Bob merge le travail d'Alice et pousse les modifs dans son dépôt distant :

```
git merge Alice/master
git push
```

Puis detruit la branche locale d'Alice :

```
git branch -d Alice/master
```

3.41 Alice se met à jour :

— ajoute le remote de Bob
— **fetche le master de Bob pour se mettre à jour :**

```
git fetch Bob master
```

— **Fusionne :**

```
git merge Bob/master
```

3.42 Alice travaille sur une branche git :

Alice doit par exemple intégrer une feature de connexion à une base de données. Elle va pour cela créer une branche *bd* dédiée à la réalisation de cette feature et se placer dedans.

3.42.1 Création et choix de la branche :

```
git checkout -b bd
```

Elle fait ensuite son travail, le teste puis :

```
git commit -am "Intégration BD"
```

3.42.2 Alice pousse sa branche sur son remote :

```
git push origin bd
```

3.43 Bob et la branche d'Alice :

3.43.1 Bob récupère la branche d'Alice :

```
git fetch Alice bd
```

3.43.2 Bob consulte la branche d'Alice :

S'il le souhaite, Bob consulte la liste des branches disponibles puis se place dans la branche d'Alice pour faire une petite revue du code de sa collaboratrice…

```
git branch -av
git checkout Alice/bd
```

3.43.3 Bob revient dans sa branche master :

```
git checkout master
```

3.43.4 Bob merge la branche d'Alice et pousse les modifs :

```
git merge Alice/bd
git push
```

3.44 Alice récupère la dernière version du master :

3.44.1 Alice fetche le master de Bob pour se mettre à jour :

```
git fetch Bob master
git merge Bob/master
```

3.44.2 Alice efface sa branche bd :

```
git branch -d bd
```

CHAPITRE 4

GIT

Tout bon développeur doit aujourd'hui savoir utiliser un système de gestion de version pour son code et pour collaborer. Git est aujourd'hui le plus répandu. Vous trouverez à la fin de ce cours un rappel des principales commandes git pour démarrer :

GIT start

et quelques commandes pour travailler à plusieurs sur un projet avec les branches git :

GIT branches

CHAPITRE 5

Références

— Manuel PHP (http ://php.net/manual/fr/)
— intro JS (https ://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/js-initiation/slides1.html#/le-langage-javascript)
— compléments JS (https ://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/js-initiation/slides2.html#/versions-de-javascript)

CHAPITRE 6

Index et recherche

— genindex
— search

Index

U

V

W

X

Z